PC magazin

Nr. 3 1990
REVISTA INDEPENDENTA
DE INFORMATICA

Ce este ADISAN?
Din istoria calculatoarelor personale (I)
Limbajul C
Concepte ale programarii orientate catre obiect
Protectia impotriva virusilor
A simula? Nimic mai simplu!

ADISAN

PC-MAGAZIN NR. 3/1990

Colectivul de redactie al revistei

La elaborarea acestui numar au colaborat:

Tiberiu Spircu, Ion Paraschiv, Adrian Popa, Irina Negru, Mircea Crutescu Octavian Paiu, Bodosi Imre, Marius Sturzoiu, Mihaela Olteanu, Mihai Trandafirescu, Mihai Unghianu, Carmen Martin, Catalina Stancioiu, Dragos Riscanu, Eugen Ionescu, Ionel Ruse, Cristian Groza, Bogdan Ciorica, Radu Ciorica, Daniela Straistaru, Ileana Straistaru, Dan Cretescu, Serban Slivinschi, Dan Balotescu, Sandu Modrescu.

Tiparul a fost executat la Tipografia Universul, 1990

Dedicatia numarului trei: "*Celor care ne-au ajutat sa ne cistigam independenta, astfel incit revista noastra sa devina ceea ce este...*"

PC-MAGAZIN™ , numarul 3/1990
Revista independenta de informatica si calculatoare, fondata de ADISAN®.

Cuprins

- Fundamente de matematica ale programarii logice in inteligenta artificiala (III) 2
- Ce este ADISAN? 9
- Exploatarea microsistemelor in limbajele Prolog si Assembler (II) 11
- Despre arhitectura microcalculatoarelor IBM PC 28
- Din istoria calculatoarelor personale (II) 30
- Limbajul C (III) 34
- Initiere in conceptele comunicatiei de date (II) 38
- Ce place la un joc pe calculator? . . . 43
- Utilizarea mouse-ului pe calculatoare personale 46
- Intre programare si joc 52
- Editoare si fonturi (II) 56
- Concepte ale programarii orientate catre obiect 60
- Ce nu se prea stie despre SPECTRUM? . . . 66
- A simula? Nimic mai simplu! 68
- Protejarea impotriva virusilor 71
- PC-CLIP. Concurs cu premii 73
- Hardware/Software. Preturi 76
- Curier editorial 78

Fundamente de matematica ale programarii logice in inteligenta artificiala (III)
Algoritmi fundamentali ai programarii logice

Adrian Negru

III.1. Unificarea

III.1.1. Principiul unificarii.

Sa ne reamintim o lege a inferentei mentionata in primul capitol si anume legea modus ponens care afirma ca daca f1 si f2 sint rbd-uri atunci din f1 si f2 ⇐ f1 se deduce f2.(¬f2 ⇐ f1 ⇔ f2:-f1 ⇔ f1 ⇒ f2). Aceasta lege devine generala sau lege modus ponens universala sub definitia:

Din regula clauza = (P ⇐ Q1,Q2,...,Qn) si faptele (Q*1,Q*2,...,Q*n) se poate deduce P1, daca o instantiere a lui clauza este P1 ⇐ Q*1,...,Q*n.

Putem, astfel, enunta principiul functional al unui interpretor logic sub forma urmatoarelor reguli algoritmice:

1) Intrare: Se pune o realizare G si un program P.

2) Iesire : Se raspunde cu DA sau NU dupa cum G este deductibila din programul P sau nu.

Rezolventul initial este G. Ciclu: daca multimea rezolvantilor nu e vida atunci:

3) Se alege o realizare Ri si o instantare clauzala a ei:

Ri ⇐ C1,...,Cn, n > 0

Se determina un nou rezolvent din:

R1,...Ri-1,C1,...,Cn,Rn + 1,...

4) Sfirsit.

Daca multimea rezolvantilor este vida raspunsul este DA, altfel NU.

Sa observam ca, in fapt, avem o forma pseudo-cod a refularii clauzale descrisa la sfirsitul capitolului II. Putem astfel determina daca un program logic produce exact tot ce ne intereseaza, putind pune in evidenta toate realizarile ce se pot deduce din program, ele constituind si multimea informatiilor acelui program logic.

Sa fixam, in continuare, citeva definitii necesare. Spunem ca un termen t1 este mai general decit t2 daca t2 este o instantiere a lui t1 dar t1 nu este instantiere pentru t2.

Spunem ca doi termeni t1 si t2 sint variante alfabetice daca t1 este instantiere pentru t2 si analog (parinte (X,Y) si parinte (Z,U) sint variante alfabetice). Un unificator pentru doi termeni este o substitutie ce face ca termenii sa devina identici, caz in care se mai spune ca ei unifica. Substitutia se numeste instantiere comuna, o proprietate duala a unificatorilor si instantarilor comune este aceea ca orice unificator determina o instantiere comuna a doi termeni si reciproc.

Numim un cel mai general unificator a doi termeni unificatorul pentru care instantierea comuna determinata este cea mai generala (instantiere comuna mai generala decit cea gasita).

Algoritmul unificarii consta tocmai in determinarea celui mai general unificator a doi termeni sau o confirmare de esec, in caz contrar.

Principial, algoritmul porneste de la doi termeni, term1, term2, producind la iesire instantierea comuna cea mai generala, in caz de reusita, sau esec, in caz contrar.

Metoda abordata de algoritm consta in furnizarea unei stive in care sint stocate toate ecuatiile rezolvante si o zona de memorie, SUBST, unde se concateneaza substitutiile ce vor forma multimea instantiere comuna a unificarii. Zona SUBST este initial goala ca de altfel si stiva in care se pune la inceput ecuatia term1 = term2.

Daca term1 = term1 (X1,X2,...,Xn) si term2 = term2(Y1,Y2,...Yn) se pun in stiva pe rind ecuatiile componentelor Xi = Yi, i = 1,n. Actiunea algoritmului consta acum in extragerea (functia pop()) din stiva a ecuatiilor Xi = Yi, i = 1,n si procesarea lor in sensul instantierii. Algoritmul inceteaza daca stiva devine goala sau daca o ecuatie Xk = Yk devine imposibil de procesat. Posibilele variante ale ecuatiilor Xi = Yi sint:

i) Xi si Yi pentru un i fixat sint constante identice sau variabile identice, caz in care una din ele este instanta a celeilalte si se proceseaza urmatoarea ecuatie Xi + 1 = Yi + 1;

ii) Xi este variabila si Yi un termen ce nu contine pe Xi. Atunci, in toata stiva se proceseaza inlocuirea (instantierea) lui Xi cu Yi.

La fel, daca Xi apare in zona SUBST, el este instantat cu Yi, adaugind zonei si substitutia {Xi | Yi},intelegind prin {X | a} ca "a" il substituie pe X, [X = a]. Reciproc se procedeaza daca Yi este variabila si Xi termen ce nu contine pe Yi. Daca procedeul continua pina la golirea stivei atunci unificatorul este in zona SUBST.

Ca exemplu sa unificam termenii:

lista ([X | [mere | Ys]])

lista ([pere | [2 | [X | ls]])

Stiva este initializata cu ecuatia:

lista ([X | [mere | Ys]]) = lista ([pere | Z | [X | ls]])

care produce ecuatiile partiale:

X = pere; [mere | Ys] = [Z | [X | ls]] care,la rindul ei,produce ecuatiile:

mere = Z; Ys = [X | ls].

In continuare,ecuatia "X = pere" este scoasa din stiva si procesata in sensul ca,variabila X nu apare in constanta pere,iar toate aparitiile lui X in stiva sint substituite cu pere.O singura ecuatie este afectata si anume:

Ys = [X | ls], care devine Ys = [pere | ls];

In continuare substitutia {X = pere} se adauga zonei SUBST (initial vida) si procedeul continua.

Urmatoarea ecuatie mere = Z instanteaza variabila Z ce nu apare in constanta mere si substitutia se adauga la zona SUBST. Iar, in ultima ecuatie, Ys = [pere | ls], cum Ys nu apare in lista [pere | ls], ecuatia se adauga asa cum este la unificator in zona SUBST si algoritmul se termina cu succes. Unificatorul este dat de :{X = pere, Z = mere, Ys = [pere | ls]}, instantarea comuna produsa de unificator fiind:

lista ([pere | [mere | [pere | ls]]]).

Procesul de unificare a doua structuri este un algoritm global, in sensul ca, orice conflict de instantare intre doua elemente anuleaza substitutiile precedente efectuate cu succes, rejectind printr-un raspuns de esec (fail) unificarea structurilor si, de asemenea, exista mai multe cai de aducere la o forma unitara a doi termeni in urma unificarii. Sa analizam incercarea de unificare a termenilor:

proprietate (vila,X) si proprietate (Y,Z).

Exista o singura unificare sigura si anume instantarea {Y = vila}.

Valorile sub care X si Z vor fi instantate sint neimportante in acest proces, singura conditie impusa lor fiind aceea a identitatii lor absolute, X = Z. Deci, nu poate fi vorba, in acest caz, de gasirea unui cel mai general unificator deoarece instantarea X = Y sau Z = X lasa in substitutie un termen variabil, deci fara o valoare definita,aceasta neimpiedicind insa ca procesul de unificare sa se realizeze totusi cu succes. Mai mult, c.m.g.u. nu este unic in sensul ca nu putem spune daca X se instanteaza cu Z sau X = Z ori invers, Z = X, pentru ca ambele instantari nu afecteaza derularea in continuare a programului ci indica doar daca predicatul trece la o valoare a corpului de program sau primeste o valoare prin instantare.

Ceea ce mai trebuie remarcat este ca, in procesul de unificare, termenii din stinga sau dreapta ce se vor unifica pot fi schimbati arbitrar ca pozitie, adica procesarea ecuatiilor Xi = Yi sau Yi = Xi produce acelasi efect de instantare, tinind cont de urmatoarele patru restrictii impuse asupra posibilelor echivalente de termeni intr-o ecuatie:

i) Cei doi termeni (membri) ai ecuatiei sint formule atomice constante [fapte]. Unificarea proceseaza o comparare a valorilor reusind, daca ele sint respectiv identice, esuind altfel.

ii) Unul din termeni este o variabila neinstantata. Unificarea instanteaza automat variabile cu termenul corespunzator, indiferent de structura acestuia.De exemplu:[X | Xs] = [a | [b | Ys]] produce instantarea corpului Xs al primei liste care este o variabila neinstantata cu lista [b | Ys] prin substitutia {Xs = [b | Ys]}.

iii) Unul din termeni este o variabila instantata. Atunci, aceasta valoare este unificata cu celalalt termen. Procesul poate fi recursiv, in sensul ca aceasta valoare poate fi ea insasi instantarea unei alte variabile; acest caz putindu-se, la o utilizare eficienta, in cazul posibilitatii minuirii de pointeri a sistemului sub care se lucreaza. Avem exemplul:

[a | X] = [[Y | a] | Z], cu substitutia a = [Y | a] = [Y | [Y | [Y | [... | [Y | a]], obtinuta prin aplicarea recursiva a regulii de n ori.

iv) Ambii termeni sint structuri cu acelasi numar de argumente caz in care se aplica algoritmul unificarii descris mai inainte si conditiile i,ii,iii. Daca structurile au un numar diferit de argumente instantarea nu are sens.

Un alt exemplu de felul cum actioneaza unificarea in ambii termeni il consideram util pentru constientizarea efectului ei in programare.

Sa presupunem ca vrem sa stabilim in ce conditii doua intreprinderi, X si Y pot, stabili un contract asupra unei anumite realizari obiect fata de care X este intreprinderea furnizoare iar Y cea beneficiara. Problema se poate formula prin:

(1) contract (X,Y,Obiect): -furnizor (X,Obiect),

-beneficiar (Y,Obiect).

Sa vedem cum unifica cele doua predicate furnizor si beneficiar sub cererea de realizare (1) contract(X,Y,Obiect) daca avem definitii in baza de date functorii:

furnizor (Electronica,utilaj (Plan financiar, calculator, [memorie, disc_2OOM]

intelegind ca intreprinderea Electronica este furnizoarea unui utilaj al carei pret este cuprins intr-un plan financiar, el fiind un calculator cu o memorie memorie, neinstantata, putind fi o variabila, ceea ce afecteaza si Plan financiar, sau pretul calculatorului, si continind in componenta un disc 2OOM sau un disc de 200 Mega bytes ce constituie o constanta valabila pentru orice utilaj .

Daca predicatul beneficiar are formulata cererea:

beneficiar(Automatica,utilaj(pret(mai_mic_1000000,mod_de_plata),CORAL,configuratie)

atunci unificarea celor doi functori sub cererea:

contract(X,Y,Obiect)?

prin substitutia:

{X = Electronica};{Y = Automatica};

{Plan financiar = pret(mai_mic_1000000,mod_plata};

{calculator = CORAL};configuratie = [memorie | disc_2OOM];

si deci:

{Obiect = utilaj(pret(mai_mic_1000000,modplata),CORAL,[memorie_disc_2OOM])}

produce ca raspuns al unificarii,realizarea:

contract(Electronica,Automatica,utilaj(mai mic 1000000,mod_plata,CORAL,[memorie | disc_2OOM])

ceea ce arata ca poate exista un posibil contract intre cele doua intreprinderi ce priveste un calculator CORAL in care nu au importanta modul de plata si dimensiunea memoriei ci singurele conditii sint ca acest calculator sa aiba un disc de 200 Mega bytes si un pret mai mic de 1.000.000lei.

III.1.3. Un algoritm al unificarii sub forma de program logic

Manipularea datelor in programarea logica isi gaseste expresia cea mai sigura prin mecanismul unificarii ce presupune atribuirea unica, trecerea parametrilor, alocarea inregistrarilor si scrierea sau citirea de cimpuri particulare ale unei inregistrari sau structuri. Sa incercam in aceste circumstante definirea unificarii a doi termeni printr-un predicat sa-l numim

unific(termen1,termen2)

realizarea lui fiind satisfacuta daca termen1 unifică cu termen2, altfel nu. Sa facem o observatie importanta si anume delimitarea stricta a doua cazuri posibile si anume acceptarea sau ignorarea unui principiu fundamental al unificarii si anume cererea ca o variabila sa nu fie unificata cu un termen ce contine acea variabila.

Satisfacerea acestei conditii este o cerinta majora a procesului de implementarea a unificarii in limbajul Prolog sau orice alt limbaj al programarii logice care cere sa se proceseze verificarea identitatii a doua variabile si nu numai posibilitatea unificarii lor, adevarata, dupa cum am vazut pentru crice doua variabile.

Programul logic care defineste clauza unific (term1,term2) mai foloseste in structura doua predicate ce pot fi interne sau nu unui sistem al programarii logice si anume predicatele functor si argument precum si un predicat derivat al predicatelor interne si anume compus(X) a carui valoare este adevarata daca X este un termen compus*. Astfel, predicatul functor. In diferitele implementari ale limbajului Prolog aceste predicate pot fi definite intern, eventual numele lor din echivalentul englezesc, traducerea fiind naturala in termenii bunului simt: functor, arg sau argument si compound(compus).

permite accesul la numele, argumentele si dimensiunea unui termen compus prin definirea lui sub forma:

functor(Termen,Nume,Dimensiune)

valoarea fiind adevarata daca Termen este un termen compus cu principalul predicat Nume si numarul de argumente dat de Dimensiune.Astfel, functor(adresa (Adi, 1Mai), 2) are valoare de adevar. Constantele le vom considera functori de dimensiune zero pentru a completa definitia.

Definirea si valoarea de adevar a acestui predicat este direct legata de baza de date a faptelor asociate, o relatie de forma functor (adresa(X,Y), proprietar, 2) esuind, procesul unificarii neavind loc intre predicatele adresa si proprietar. De asemenea, predicatul functor nu este realizat daca argumentul Dimensiune este atom sau termen compus si, mai mult, vom semnala un mesaj de eroare (dar nu de esec!) daca o intrebare implicind functor (de exemplu functor (X,Y,3)?) produce o infinitate de solutii.

Urmeaza definitia, aproape naturala, a predicatului argument, sau arg pe scurt, si anume:

arg(Numar,Termen,Argument)

care are valoarea de adevar daca Argument este al Numar argument al termenului Termen. Astfel: arg(2,adresa(Adi,1Mai),1Mai)are valoare de adevar iar cererea arg(N,adresa(X,1Mai),X)produce raspunsul{X = Adi,N = 1}daca unicul fapt in baza de date este adresa(ADi,1Mai) sau (N = 2) daca intrebarea se pune sub forma:arg(N,adresa(-,1Mai),1Mai)?

De asemenea, avem urmatorul raspuns la intrebarea: arg(1,adresa(X,1Mai),Adi)?-R:{X = Adi};

esecul acestui predicat producindu-se daca componentele sale unifica in baza de date a faptelor sau daca primul argument este un atom si nu un intreg [arg (2, adresa (Adi, 1Mai), Mar; arg("floare", adresa (Adi, 1Mai), Adi)] iar eroare fiind semnalata in cazul unei ambiguitati de forma arg(2,X,Z)?

Cu aceste definitii sa elaboram primul program logic al unificarii si anume acela care nu tine seama de unificare unei variabile cu un termen ce contine variabila:

Program-de-unificare-1

(1) unific(X,Y):- var(X), var(Y),X = Y.

(2) unific(X,Y):- var(X), nonvar(Y),X = Y.

(3) unific(X,Y):- var(Y), nonvar(X),Y = X.

(4) unific(X,Y):- nonvar(X), nonvar(Y), constant(X), constant(Y), X = Y.

(5) unific(X,Y): - nonvar(X), nonvar(Y), compus(X), compus(Y), termen_unific(X,Y).

(6) termen_unific(X,Y): - functor(X,Nume,Dimens),

- functor(Y,Nume,Dimens),

- arg-unific(Dimens,X,Y).

(7) arg_unific(Dimens,X,Y):-Dimens0,arg_unific(Dimens,X,Y),

- Dimens1 is Dimens-1,

- arg_unific (Dimens1,X,Y).

(8) arg_unific (0,X,Y).

(9) arg_unific (Dimens,X,Y):-arg(Dimens,X,ArgX),

- arg(Dimens,Y,ArgY),unific(ArgX,ArgY).

(10) compus(X):- atomic(X).

(11) compus(X):- struc(X).

(12) compus(X):- atom(X).

Sa observam ca programul prezinta, prin clauzele 1-5, posibilitatea de unificare a doi termeni din punct de vedere al "tipului" lor, discutat anterior. Se introduce apoi un nou predicat termen_unific care cerceteaza daca doi termeni compusi pot unifica (daca au acelasi nume si acelasi numar de argumente) dupa care proceseaza prin predicatul arg_unific descompunerea descrisa in principiul unificarii in ecuatii de tip Xi = Yi, i = 1,2,...Dimens, procesind unificarea fiecaruia dintre ele daca Xi,Yi sint termeni simpli prin apelul recursiv al lui unific (ArgX, ArgY), care proceseaza unificarea daca termenii satisfac una din conditiile 1-5 sau executa apelul lui termen-unific daca, din nou, ArgX si ArgY sint compusi.

Sa exemplificam actiunea predicatului arg-unific pentru:

X = [a | [b | c]]; Y = [Z | Zs].

Avem:

arg_unific (2,[a | [b | c]],[Z | Zs]) ne da

arg(2,[a | [b | c]],ArgX) cu solutia ArgX = [b | c]

iar arg(2,[Z | Zs],ArgY) cu solutia ArgY = [Zs]

unde aplicind unific(ArgX, ArgY)se obtine unific([b | c], Zs) cu solutia{Zs = [b | c]}. Urmeaza decrementarea lui Dimens = 2 la Dimens1 = Dimens-1 = 2-1 = 1 si aplicarea lui arg unific(1,[a | [b | c],[Z | Zs]) ce ne da:ArgX = a; ArgY = Z si unificarea unific(a,Z) cu solutia pentru substitutia {Z = a}.

Sa extindem programul anterior introducind cererea ca o variabila sa nu unifice cu un termen ce contine variabila si, mai mult, daca doua variabile se intilnesc in procesul de unificare ele sa unifice numai daca sint identice.

Orice studiu de programare logica furnizeaza un predicat intern pentru testul identitatii a doua variabile prin,identic(X,Y) = " = = " sau identic (X,Y) are loc daca X = = Y,adica testul asupra lui X si Y reuseste daca X si Y sint variabile identice sau constante identice sau X si Y sint structuri cu acelasi functor si orice ecuatie partiala Xi = = Yi are loc.Testul X = = "a" sau Y = = 2 produc esec in comparatie cu echivalenta atributiva " = " care instanteaza X cu "a" si Y cu 2.Negatia operatorului " = = " este tot un predicat intern dat prin operatorul "\ = = ".Testul 10\ = = 10 esueaza in timp ce 10\ = = 9 are succes.

Programul anterior se va completa cu un predicat nou,nu apartine(Termen1,Termen2),care are succes in momentul in care variabila Termen 1 nu apare in componenta termenului Termen2.

Forma completa a programului de unificare implementabila sub orice compilator logic o prezentam in continuare.

Program_de_unificare_2

(1) unific(X,Y):- var(X), var(Y), X = Y.

(2) unific(X,Y):- var(X), nonvar(Y), nu-apartine(X,Y), X = Y.

(3) unific(X,Y):- var(Y), nonvar(X), nu apartine(Y,X), Y = X.

(4) unific(X,Y):- nonvar(X), nonvar(Y), constant(X), constant(Y), X = Y.

(5) unific(X,Y): -nonvar(X), nonvar(Y), compus(X), compus(Y), termen_unific(X,Y).

(6) termen_unific (X,Y):- functor(N,Nume,Dimens), - functor(Y,Nume,Dimens),

-arg_unific(Dimens,X,Y).

(7) nu_apartine(X,Y):- var(Y),X\ = = Y.

(8) nu-apartine(X,Y):- nonvar(Y),constant(Y).

(9) nu-apartine(X,Y):- nonvar(Y), compus(Y), functor(Y,Nume,Dimens),

- nu_apartine1(Dimens,X,Y).

(10) nu_apartine1(Dim,X,Y):-Dim0,arg(Dim,Y,argY).

- nu_apartine1(Dim1,X,Y).

(11) nu_apartine1(0,X,Y).

(12) arg_unific (Dim,X,Y) - este prezentat in programul anterior.

Distingem cele doua predicate nu_apartine si nu_apartine1 care sint definite clauza1 astfel:

a) Clauzele 7-9 impun realizarea unificarii daca Y este variabila si X neidentic cu Y,atunci X nu figureaza in Y (7) iar X nu figureaza in Y pentru Y constanta. Daca Y este un termen compus atunci se apleaza recursiv predicatul nu_apartine1 care este realizat daca, pe rind, se verifica pentru fiecare argument Yi, al lui Y, (i = 1,2,...,Dim) unde Dim este numarul de argumente al lui Y) ca X nu figureaza in Yi sau are loc realizarea nu_apartine(X,Yi).

Ce este "ADISAN"?

Adrian Negru si Alexandru Babin

Constituirea legala a intreprinderii "ADISAN" este rezultatul eforturilor noastre de a crea formula lucrativa a activitatii "Asociatiei Specialistilor in Baze si Banci de Date", grupare profesionala pur stiintifica.

"ADISAN" este o companie independenta orientata in domeniul microinformaticii care isi propune sa-si aduca aportul in procesul de informatizare al tarii. Pornind de la premisele dezvoltarii unei culturi informatice, s-au pus bazele infiintarii unei edituri pur de specialitate in domeniul informatic, testul initial de piata fiind revista cu aparitie "aproape" lunara "PC-Magazin" care a trebuit si inca trebuie sa suporte bariere, birocratie si imobilism pina a deveni ceea ce noi speram sa devina. Nefiind inca suficient de bine organizati din punct de vedere administrativ/comercial a trebuit sa ne confruntam cu dificultati majore in distributia revistei in tara si poate multa vreme va mai fi asa. Oricum, librariile bucurestene au inteles interesul de care se bucura revista si au preluat o mare parte a tirajului. Incercam, prin distribuitori, voluntari deocamdata, sa facem ca revista sa ajunga in orasele mari ale tarii.

Mai mult, pina la a ajunge in situatia de a beneficia de propria noastra baza materiala care sa ne asigure independenta lucrativa, a trebuit sa apelam pe ici pe colo la "milostenia" publica.

In sfirsit insa, cu 10 microcalculatoare AST/286, o imprimanta Laser, doua retele (una Novell si una 3 + Comm + Presentation Manager + OS/2) ne cistigam incet, incet independenta care sa ne asigure afirmarea scriptica a potentei noastre stiintifice. Beneficiul direct se afirma in posibilitatea de ridicare a nivelului tipografic al revistei si punerea in viata a proiectului de editare a unui numar de brosuri informatice pentru incepatori si avansati care vor fi oferite de libraria "ADISAN" a carei aparitie o apreciem posibila incepind cu luna august-septembrie, problemele fiind legate de spatiu comercial si potential financiar care, cu perseverenta si credinta, speram sa le rezolvam. Intentionam, luind in consideratie cerintele multor cititori, sa editam o revista dedicata utilizatorilor de calculatoare Spectrum, Commodore, HC, Cobra, ca si un supliment hardware dedicat celor pasionati de electronica si tehnologie informatica. Dincolo de activitatea editoriala, ADISAN isi propune sa dezvolte pachete software care sa satisfaca variatele nevoi ale pietii interne sau externe. Astfel, in acest moment, ADISAN si-a pus masca pe urmatoarele produse software:

- LIBARCH, Sistem de baza de date care permite gestiunea inventarelor de bunuri materiale ale unei intreprinderi;

- gRAPHIX, pachet specializat in gestiunea si optimizarea activitatilor economice;
- SYMBOLIC, biblioteca matematica integrata cu posibilitati de grafica in 2D si 3D ca si analiza bazata pe calculul formal al ecuatiilor diferentiale liniare;
- EXP, sistem expert de interfata intre utilizator si o baza de date;
- RUBIK, jocul "cubul Rubik" rezolvat pe calculator.

Toate aceste pachete sint scrise in cod C si Prolog beneficiind de interfata utilizator de tip WINDOWS.

In sfera serviciilor nu ne-am propus inca sa oferim solutii ci numai sa raspundem cererilor de asistenta si maintenance soft/hard.

Astfel, la cererea Uniunii Scriitorilor din Romania, am organizat pe calculator alegerile Adunarii Generale a acesteia, dezvoltind, in acest sens, un pachet in cod C numit ELECT care, gestionat in retea, a permis centralizarea introducerii si validarii buletinelor de vot. Avem, alaturat, scrisoarea de multumire adresata noua de presedintele nou ales al Uniunii, poetul Mircea Dinescu.

Ca marea majoritate a intreprinderilor, emanatie a "liberei initiative", ne-am confruntat si ne confruntam cu probleme legate de asigurarea unui spatiu stabil in care sa functionam. In intelegerea greoaie ca lumea se schimba si fenomenul de tranzienta al locului de munca este mai vizibil ca oricind si, oricit este de greu de acceptat, sita de cernere a valorilor incepe sa functioneze numai pe probitate profesionala si nu pe alte considerente. Mai mult, este necesar sa se inteleaga ca informatica se situeaza in sfera productiei materiale si nu este o forma administrativa de tip "TESA".

UNIUNEA SCRIITORILOR
DIN
ROMANIA

Nr. 1124 din 3 mai 1990

CATRE

ASOCIATIA SPECIALISTILOR IN BAZE SI BANCI DE DATE "ADISAN"
Str. Turda nr. 114, Bloc 35, Sc. B, Ap. 71,
Bucuresti

Avem placerea sa va adresam multumirile noastre pentru modul in care ati organizat si desfasurat lucrarile de numarare computerizata a voturilor la *Adunarea Generala a Uniunii Scriitorilor din Romania*, care s-a desfasurat in perioada 17-21 aprilie 1990.

Ne exprimam speranta ca si in viitor raporturile noastre de colaborare vor continua in mod fructuos.

PREȘEDINTE,
Mircea Dinescu

EXPLOATAREA MICROSISTEMELOR IN LIMBAJELE PROLOG SI ASSEMBLER (II)

Adrian Negru

FUNCTIA 01H

Citeste tastatura ču ecou

Apel

AH = 01H

Intoarcere rezultat

Caracter tiparit

Functia 01H asteapta ca un caracter sa fie citit de la tastatura si tipareste caracterul pe suportul de iesire asignat (ecran), de asemenea, punind valoarea sa in registrul AL.

```
Macro defineste:

citeste_tast_ecou

macro

mov  ah,01H

int  21H

endm
```

Exemplu Prolog:

citeste_tipareste:

AX = 1 X 256 /*mut in AH codul 01*/

Sa facem observatia importanta ca TURBO PROLOGUL ne pune la dispozitie predicatul:

bitand(X,Y,Z)

definit prin Z: = X AND Y avind ca efect operatia de and aritmetic intre variabilele X si Y si predicatele:

bitleft(X,N,Y),

bitright(X,N,Y)

cu actiunea

Y: = shl (shr) N sau, daca deplasam bitii din reprezentarea binara a lui X cu N pozitii la stinga (dreapta) obtinem pe Y.

In continuare pentru fiecare functie vom prezenta rutina de apel scrisa in limbaj de asamblare urmata de versiunea sa Prolog care, daca apare explicit in paginile acestui capitol, o vom gasi in capitolele urmatoare insotita de motivatiile necesare cu comentarea rezultatului intors de intrerupere.

FUNCTIA 02H

```
Afisez_char

DISPLAY_CHAR macro caracter

mov  dl,caracter

mov  ah,02H

int  21H

endm
```

Afisez_Car (C):

bitand(DX,C,DX)

DX = C

AX = $0200

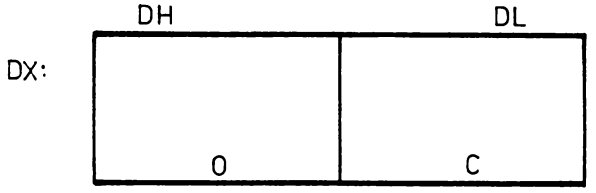
bios($21,reg(AX,-,-,DX,-,-,-,-),-)

Apelul afisez_char (A) va scrie pe ecran caracterul A.

```
FUNCTIA 03H
Intrare_aux:
mov  ah,03H
int  21H
endm
Intrare_aux:
AX = $0300
bios($21,reg(AX,-,-,-,-,-,-,-),reg(C
,-,-,-,-,-,-,-))
CAR = C, write(CAR).
```

Functia de 03H citeste caracter in AL de pe un suport de intrare auxiliar. Rutina Prolog tipareste acest caracter. Se poate retine acest caracter si intr-un fisier utilizator.

In rutina auxiliara faptul ca am depus in DL octetul inferior al registrului DX, caracterul C s-a facut prin atribuirea lui DX a lui C care este inscris in DL si stergerea octetului superior DH a registrului prin apelul la bitand(DX,C,DX) configuratia registrului DX fiind acum:



Prologul ne mai pune la dispozitie un predicat pentru setarea unui registru la valoarea 0 si anume:

bitxor(X,Y,Z)

ce are ca efect operatia Z = X xor Y

Astfel, bitxor(A,A,B) da lui B valoarea A xor A, adica 0.

Rutina se mai putea scrie modificat.

afisez_car (C):

bitxor(DX,DX,DX),

DX = C, /*caracterul se inscrie in DL*/

AX = $0200,...

Urmatoarele macro-uri de utilizare a functiilor lui (INT 21H) sint (acolo unde nu vor interveni in capitolele urmatoare) urmate de secventele lor de utilizare scrise in Prolog.

```
FUNCTIA 04H
iesire_aux macro
mov  ah,04H
int  21H
endm
iesire_aux: - AX = $0400
bios($21,reg(AX,-,-,-,-,-,-,-),-)
```

```
FUNCTIA 05H
tip_car macro caracter
mov  dl,caracter
mov  ah,05H
int  21H
endm
tip_car (C): - DX = C,
bitand(DX,C,DX)
AX = $0500,
bios($21,reg(AX,-,-,DX,-,-,-,-),-)
```

```
FUNCTIA 06H
I/E_cons macro switch
mov  dl,switch
mov  ah,06H
int  21H
endm
I/E_cons (Switch): -
bitxor(DX,DX,DX),
DX = Switch, AX = $0600,
bios($21,reg(AX,-,-,DX,-,-,-,-),-)
```

```
FUNCTIA 07H
intr_cons macro
mov  ah,07H
int  21H
endm
Intr_cons: - AX = $0700,
bios($21,reg(AX,-,-,-,-,-,-,-),reg(C
,-,-,-,-,-,-,-)),
write(C)
```

Functia citeste un caracter de la tastatura sau de pe suportul de intrare si il pune in AL. Este utila in accesul cu parola intr-un program. Rutina Prolog afiseaza caracterul.

```
FUNCTIA 08H
cit_cons macro
mov  ah,08H
int  21H
endm
cit_cons: - AX = $0800
bios($21,reg(AX,-,-,-,-,-,-,-),reg(C
,-,-,-,-,-)
```

Functia citeste un caracter de la consola si-l depune in AL.

```
FUNCTIA 09H
display macro sir
mov  dx,offset,sir
mov  ah,09H
int  21H
endm
display(SIR): - ptr_word(SIR,DX,DX)
AX = $0900,
bios($21,reg(AX,_,_,DX,_,_,DX,_),_)
```

Rutina afiseaza un sir de caractere SIR ce se termina cu simbolul S. Registrul DX contine deplasamentul sirului fata de DS si se obtine prin apelul ptr_word cu specificarea sirului.

```
FUNCTIA 0BH
stare_con macro
mov  ah,08H
int  21H
endm
stare_con: - AX = $0B00,
bios($21,reg(AX,-,-,-,-,-,-,-),reg(C
,-,-,-,-,-,-,-))
egal(C,0), stare_con.
stare_con: - egal(C,$00FF), write
("s-a terminat caracter").
```

Rutina testeaza starea tastaturii si intoarce in AL coloana 0 daca nu s-a tastat nici un caracter si valoarea FF daca s-a tastat ceva.

```
FUNCTIA 0CH
citire_cu_salvare
mov  al,switch
mov  ah,0CH
int  21H
endm
```

```
FUNCTIA 0DH
reset_disk macro
mov  ah,0DH
int  21H
endm
reset_disk: - AX = $0D00,
bios($21,reg(AX,-,-,-,-,-,-,-),-)
```

Functia reseteaza discul, neactualizind directorii decit pentru fisierele inchise.

```
FUNCTIA 0EH
select_disk macro disk
mov  dl,disk-65
mov  ah,0EH
int  21H
endm
select_disk ("A"): - DX = 0, AX =
$0E00,
bios($21,reg(AX,-,-,DX,-,-,-,-),-)
```

In DL se pune numarul logic al unitatii de discuri (0 = A, 1 = B etc.) iar AL intoarce numarul unitatilor logice.

```
FUNCTIA 0FH
open macro nume_fis
mov  dx,offset nume_fis
mov  ah,0FH
int  21H
endm
open (Nume_fis): - ptr_word
(Nume_fis,DS,DX),
AX = $0F00,
bios($21,reg(AX,-,-,DX,-,-,DS,-),reg
(ST,-,-,-,-,-,-,-)),
egal(ST,0),
write (Nu exista fisier cu numele
Nume_fis)
```

Functia deschide un fisier cu Nume_fis si intoarce in AL valoarea 0 daca acesta nu exista.

Daca fisierul este ascuns sau are atribute de sistem atunci AL contine valoarea FF.

```
FUNCTIA 10H
close macro nume_fis
mov  dx,offset nume_fis
mov  ah,10H
int  21H
endm
```

Prezentam, in continuare, celelalte apeluri de functii si forma lor de apel in limbaj de asamblare. Scrierea sub forma de rutine Prolog este imediata in virtutea exemplelor anterioare si ramine ca un exercitiu. Multe dintre aceste rutine le vom folosi mai departe, in capitolele dedicate tastaturii, unitatii grafice, unitatilor de discuri si celorlalte periferice unde vor aparea programe Prolog complexe de utilizarea lor.

Mai mult, utilitatea scrierii lor sub forma de macro-uri in limbaj de asamblare se va dovedi in momentul in care prezentam inferfata dintre Prolog si alte limbaje de programare, ele raminind a fi incorporate in programe Prolog chiar sub forma data in limbaj de asamblare.

```
FUNCTIA 11H
caut_prim macro nume_fis
mov  dx,offset nume_fis
mov  ah,11H
int  21H
endm
```

Functia cauta existenta fisierului nume_fis pe disc si intoarce in AL valoarea 0 daca gaseste directorul ce contine fisierul, altfel FFH.

```
FUNCTIA 12H
caut_alt macro nume_fis
mov  dx,offset nume_fis
mov  ah,12H
int  21H
endm
```

Functia cauta existenta fisierului nume_fis; dupa eventual o prima cautare permitind, daca numele fisierului nu este specificat, determinarea numarului de fisiere existente pe un disc.

```
FUNCTIA 13H
sterge macro nume_fis
mov  dx,offset nume_fis
mov  ah,13H
int  21H
endm
```

Este sters fisierul nume_fis de pe disc, iar daca el nu exista in AL se intoarce valoarea FF.

```
FUNCTIA 14H
cit_seq macro nume_fis
mov  dx,offset nume_fis
mov  ah,14H
int  21H
endm
```

Citeste o inregistrare din fisierul nume_fis si AL contine codul procesorului astfel: 0 - citire completa, 1 - nu sint date in fisier, 2 - nu este loc la adresa de transfer a discului pentru citirea unei inregistrari, 3 - detectat sfirsit de fisier fara citire completa a ultimei inregistrari.

```
FUNCTIA 15H
scriere_seq macro nume_fis
mov  dx,offset nume_fis
mov  ah,15H
int  21H
endm
```

Codul intors in AL:

00H - scriere completa;

01H - disc plin;

02H - adresa de transfer disc prea mica.

```
FUNCTIA 16H
creaza macro nume_fis
mov  dx,offset nume_fis
mov  ah,16H
int  21H
endm
```

Sistemul cauta un fisier existent nume_fis iar daca nu gaseste deschide un fisier la adresa DS:DX. Daca se incearca crearea unui fisier existent, MS-DOS-ul il sterge si creaza unul nou, gol.

```
FUNCTIA 17H
redenumeste macro nume_vechi,
nume_nou
mov  dx,offset nume_vechi, nume_nou
mov  ah,17H
int  21H
endm
```

Schimba numele unui fisier cu deplasamentul (unui FCB cu numarul drive-ului si numele fisierului vechi, nume_vechi in el urmat de numele nume_nou la deplasamentul 11H) in DX. AL contine 0 daca nu a gasit fisierul.

```
FUNCTIA 19H
disk_curent macro
mov  ah,19H
int  21H
endm
```

Intoarce in AL drive-ul curent (0 = A, 1 = B etc.).

```
FUNCTIA 1AH
set_dta macro bufer
mov  dx,offset bufer
mov  ah,1AH
endm
```

Functia stabileste adresa de transfer disc.

```
FUNCTIA 1BH
inf_disc macro
mov  ah,1BH
int  21H
endm
```

Se dau informatii despre disc astfel: AL contine numarul de sectoare dintr-un cluster, CX contine numarul de octeti dintr-un sector, DX contine numarul clusterelor din disc iar BX deplasamentul primului octet al FAT-ului ce identifica tipul discului din unitate (FF - disc dubla densitate, 8 sectoare pe pista, 40 piste pe fata) (F8 - daca este hard disc).

```
FUNCTIA 1CH
drive_data macro drive
mov  dl,drive
mov  ah,1CH
int  21H
endm
```

Intoarce informatii despre discul situat intr-un drive specificat cu semnificatia registrilor ca mai sus.

```
FUNCTIA 21H
cit_ram macro nume_fis
mov  dx,offset nume_fis
mov  ah,21H
int  21H
endm
```

Citeste inregistrarea aratata de cimpul de inregistrare relativ (deplasament 21H) al unui fisier cu deplasamentul nume_fis in DX. AL intoarce valorile 0 - citire reusita, 1 - EOF - lipsa date, 2 - citire abandonata, 3 - EOF - citire incompleta a ultimei inregistrari care a fost completata cu valori 0.

```
FUNCTIA 22H
scrie_ran macro nume_fis
mov  dx,offset nume_fis
mov  ah,22H
int  21H
endm
```

Scrie inregistrarea indexata la deplasamentul 21H, AL continind valorile: 0 - scriere completa, 1 - disc plin, 2 - abandon.

```
FUNCTIA 23H
fis_size macro nume_fis
mov  dx,offset nume_fis
mov  ah,23H
int  21H
endm
```

Intoarce marimea fisierului ce se gaseste astfel: in blocul de control al fisierului la nume_fis [33] se gaseste adresa unde se afla lungimea unei inregistrari iar nume_fis [14] se afla adresa la care se gaseste numarul de inregistrari.

```
FUNCTIA 24H
set_relativ_inreg macro nume_fis
mov  dx,offset nume_fis
mov  ah,24H
int  21H
endm
```

Seteaza cimpul de inregistrare relativ (21H) la adresa de fisier specificata in cimpul blocului curent (deplasament 0C) si cimpul inregistrarii curente (deplasament 20H).

```
FUNCTIA 25H
set_vector macro intrerupere,
rutina_start
mov  al,intrerupere
mov  dx,offset rutina_start
mov  ah,25H
int  21H
endm
```

Seteaza adresa in tabela de intreruperi a unei intreruperi utilizator cu numarul in AL si capul rutinei ce o descrie la deplasamentul rutina_start in DX fata de DS.

```
FUNCTIA 26H
creaza_PSP macro seg_addr
mov  dx,offset seg_addr
mov  ah,26H
int  21H
endm
```

Creaza un nou PSP la adresa segmentului, continuta in DX.

```
FUNCTIA 27H
ran_bloc_cit macro
nume_fis,nr,rec_size
mov  dx,offset nume_fis
mov  cx,nr
mov  word ptr nume_fis[14],rec_size
mov  ah,27H
int  21H
endm
```

Citeste un numar de nr inregistrari dintr-un fisier nume_fis. AL intoarce 0 - pentru citire completa, 1 - EOF - fara data in inregistrare, 2 - abandon.

```
FUNCTIA 28H
ran_bloc_scrie macro
nume_fis,nr,rec_size
mov  dx,offset nume_fis
mov  cx,nr
mov  word ptr nume_fisÊ14Ê,rec_size
mov  ah,28H
int  21H
endm
```

Scrie un numar de nr inregistrari intr-un fisier nume_fis. AL intoarce 00H pentru citire completa, 01H pentru disc plin, 02H pentru sfirsit de segment.

```
FUNCTIA 29H
parse macro string,nume_fis
mov  si,offset string
mov  di,offset nume_fisier
push es
push ds
pop  es
mov  al,0FH
mov  ah,29H
int  21H
pop  es
endm
```

Analizeaza sintaxa unui sir string pentru un fisier cu forma nume disc: nume_fisier.extensie. Daca sirul contine un nume valid se creaza un fisier nedeschis nume_fis la adresa ES:DI.

```
FUNCTIA 2AH
cit_date macro
mov  ah,2AH
int  21H
endm
```

Citeste data din ceasul intern al calculatorului.

Vom lista, in continuare, si celelalte macro-uri in limbaj de asamblare ale functiilor intreruperii 21H

fara a insista asupra descrierii lor, unele din ele fiind folosite in capitolele ce urmeaza, altele insa fiind interesante doar pentru programatorii de sistem.

```
FUNCTIA 2BH
set_data macro an,luna,zi
mov  cx,an
mov  dh,luna
mov  dl,zi
mov  ah,2BH
int  21H
endm
```

Scrie o noua data in ceasul intern al calculatorului.

```
FUNCTIA 2CH
cit_ora macro
mov  ah,2CH
int  21H
endm
```

Citeste ora din ceasul calculatorului.

```
FUNCTIA 2DH
set_ora macro
ora,minute,secunde,sutimi
mov  ch,ora
mov  cl,minute
mov  dh,secunde
mov  dl,sutimi
mov  ah,2DH
int  21H
endm
```

Scrie o noua ora in ceasul calculatorului.

```
FUNCTIA 2EH
verific macro switch
mov  al,switch
mov  ah,2EH
int  21H
endm
```

```
FUNCTIA 2FH
cit_dta macro
mov  ah,2FH
int  21H
endm
```

```
FUNCTIA 30H
cit_version macro
mov  ah,30H
int  21H
endm
```

Scrie in AL versiunea sistemului de operare MS-DOS.

```
FUNCTIA 31H
continua_proces macro
return_cod,ultim_octet
mov  al,return_cod
mov  dx,offset ultim_octet
mov  cl,4
shr  dx,cl
inc  dx
mov  ah,31H
int  21H
endm
```

Mentine un proces in executie.

```
FUNCTIA 33H
ctrl_c macro actiune stare
mov  al,actiune
mov  dl,stare
mov  ah,33H
int  21H
endm
```

Seteaza sau citeste starea flagului CTRL+C. Daca AL intoarce 0, DL contine starea on sau off a lui CTRL-C. Daca AL este 1, valoarea lui DL specifica starea ce trebuie setata (1 = on; 0 = off).

```
FUNCTIA 35H
get_vector macro intrerupere
mov  al,intrerupere
mov  ah,35H
int  21H
endm
```

```
FUNCTIA 36H
get_disk_spatiu macro drive-ul
mov  dl,drive-ul
mov  ah,36H
int  21H
endm
```

```
FUNCTIA 37H
get_tara macro tara,bufer
local     gt_01
mov  dx,offset bufer
mov  ax,tara
cmp  ax,0FFH
jl   gt_01
mov  al,0FFH
mov  bx,tara
gt_01:    mov ah,38H
int  21H
endm
```

```
FUNCTIA 38H
set_tara macro tara
local     st_01
mov  dx,0FFFFH
mov  ax,tara
cmp  ax,0FFH
jl   st_01
mov  al,0FFH
mov  bx,tara
st_01:    mov  ah,38H
int  21H
endm
```

```
FUNCTIA 39H
creez_dir macro path
mov  dx,offset path
mov  ah,39H
int  21H
endm
```

Rutinele Prolog echivalente sint descrise in capitolul V al partii a doua.

```
FUNCTIA 3AH
sterg_dir macro path
mov  dx,offset path
mov  ah,3AH
int  21H
endm
```

```
FUNCTIA 3BH
schimb_dir macro path
mov  dx,offset path
mov  ah,3BEH
int  21H
endm
```

```
FUNCTIA 3CH
creez_handle macro path,atribut
mov  dx,offset path
mov  cx,atribut
mov  ah,3CH
int  21H
endm
```

Toate echivalentele Prolog ale rutinelor de lucru cu handlere de fisiere ca si filozofia lor de exploatare sint pe larg analizate in capitolul V destinat fisierelor disc (functiile 3CH-43H).

```
FUNCTIA 3DH
deschide_handle macro path,acces
mov  dx,offset path
mov  al,acces
mov  ah,3DH
int  21H
endm
```

```
FUNCTIA 3EH
inchide_handle macro handler
mov  bx,handler
mov  ah,3EH
int  21H
endm
```

```
FUNCTIA 3FH
citeste_handle macro
handler,bufer,nr_octeti
mov  bx,handler
mov  dx,offset bufer
mov  cx,nr_octeti
mov  ah,3FH
```

```
int  21H
endm



FUNCTIA 40H
citeste_handle macro
handler,bufer,nr_octeti
mov  bx,handler
mov  dx,offset bufer
mov  cx,nr_octeti
mov  ah,40H
int  21H
endm



FUNCTIA 41H
sterg_fisier macro path
mov  dx,offset path
mov  ah,41H
int  21H
endm



FUNCTIA 42H
mut_pointer macro
handle,sus,jos,metoda
mov  bx,handler
mov  cx,sus
mov  dx,jos
mov  al,metoda
mov  ah,42H
int  21H
endm



FUNCTIA 43H
schimb_mode macro
path,actiune,atribut
mov  dx,offset path
mov  al,actiune
mov  cx,atribut
mov  ah,43H
int  21H
endm
FUNCTIA 4400H,01H
IOCTL_data macro cod,handler
mov  bx,handler
mov  al,cod
mov  ah,44H
int  21H
endm



FUNCTIA 4402H,03H
IOCTL_car macro cod,handler,bufer
mov  bx,handler
mov  dx,offset bufer
mov  al,cod
mov  ah,44H
int  21H
endm



FUNCTIA 4404H,05H
IOCTL_STATUS macro cod,drive,bufer
mov  bl,drive
mov  dx,offset bufer
mov  al,cod
mov  ah,44H
int  21H
endm



FUNCTIA 4406H,07H
IOCTL_block macro cod,handler
mov  bx,handler
mov  al,cod
mov  ah,44H
int  21H
endm



FUNCTIA 4408H
IOCTL_schimb macro drive drivri
mov  bl,drive
mov  al,08H
mov  ah,44H
int  21H
endm



FUNCTIA 4409H
IOCTL_RBLOCK macro drive
mov  bl,drive
mov  al,09H
mov  ah,44H
int  21H
endm



FUNCTIA 440AH
```

```
IOCTL_rhandler macro handler
mov  bx,handler
mov  al,0AH
mov  ah,44H
int  21H
endm



FUNCTIA 440BH
IOCTL_reluare macro
reluari,asteptare
mov  bx,reluari
mov  cx,asteptare
mov  al,0BH
mov  ah,44H
int  21H
endm



FUNCTIA 440CH
generic_IOCTL_handler macro
handler,functie,categorie,bufer
mov  ch,05H
mov  cl,functie
mov  dx,offset bufer
mov  bx,handler
mov  ah,44H
int  21H
endm



FUNCTIA 440DH
generic_IOCTL_block macro
drive_num,functie,categorie,parm_blk

mov  ch,08H
mov  cl,functie
mov  dx,offset parm_blk - 1
mov  bx,drive_num
mov  ah,44H
mov  al,0DH
int  21H
endm



FUNCTIA 440EH
IOCTL_citeste_harta_drive macro
logic_nr_drive
mov  bx,logic_nr_drive
mov  ah,44H
mov  al,0EH
int  21H
endm



FUNCTIA 440FH
IOCTL_setez_harta_drive macro
logic_nr_drive
mov  bx,logic_nr_drive
mov  ah,44H
mov  al,0FH
int  21H
endm



FUNCTIA 45H
XDUP macro handler
mov  bx,handler
mov  ah,45H
int  21H
endm



FUNCTIA 46H
XDUP2 macro handle1,handle2
mov  bx,handle1
mov  cx,handle2
mov  ah,46H
int  21H
endm



FUNCTIA 47H
citeste_dir macro drive,bufer
mov  dl,drive
mov  si,offset bufer
mov  ah,46H
int  21H
endm



FUNCTIA 48H
aloc_memorie macro nr_octeti
mov  bx,nr_octeti
mov  cl,4
shr  bx,cl
inc  bx
mov  ah,48H
int  21H
endm
```

```
FUNCTIA 49H
eliberez_memorie macro seg_adresa
mov  ax,seg_adresa
mov  es,ax
mov  ah,49H
int  21H
endm
```

```
FUNCTIA 4AH
set_block macro ultim_octet
mov  bx,offset ultim_octet
mov  cl,4
shr  bx,cl
add  bx,17
mov  ah,4AH
int  21H
mov  ax,bx
shl  ax,cl
mov  sp,ax
mov  bp,sp
endm
```

```
FUNCTIA 4B00H
exec macro path,comanda,parametrii
mov  dx,offset path
mov  bx,offset parametrii
mov  ptr parametriiÊ02HÊ,offset
comanda
mov  word ptr parametrii[04H],cs
mov  word ptr parametrii[06H],ch
mov  word ptr parametrii[08H],es
mov  word ptr parametrii[0AH],ch
mov  word ptr parametrii[0CH],es
mov  al,0
mov  ah,4BH
int  21H
endm
```

```
FUNCTIA 4B03H
exec_OVL macro
path,parametrii,seg_adresa
mov  dx,offset path
mov  bx,offset parametrii
mov  parametrii,seg_adresa
mov  parametrii[02H],seg_adresa
mov  al,3
mov  ah,4BH
int  21H
endm
```

```
FUNCTIA 4CH
end_proces macro cod_retur
mov  al,cod_retur
mov  ah,4CH
int  21H
endm
```

```
FUNCTIA 4DH
asteapta macro
mov  ah,4DH
int  21H
endm
```

```
FUNCTIA 4EH
primul_fisier macro path,atribut
mov  dx,offset
mov  cx,atribut
mov  ah,4EH
int  21H
endm
```

Functiile 4E-56H sint descrise mai pe larg in capitolul V.

```
FUNCTIA 4FH
urmatorul_fisier macro
mov  ah,4FH
int  21H
endm
```

```
FUNCTIA 54h
verific macro
mov  ah,54H
int  21H
endm
```

```
FUNCTIA 56H
numesc_fisier macro
path_vechi,path_nou
mov  dx,offset path_vechi
push ds
pop  es
mov  di,offset path_nou
mov  ah,56H
int  21H
```

```
endm


FUNCTIA 57H
cit_set_data_ora macro
handle,actiune,ora,data
mov  bx,handle
mov  al,actiune
mov  ax,word ptr ora
mov  dx,word ptr data
mov  ah,57H
int  21H
endm


FUNCTIA 58H
aloc_start macro cod,strategie
mov  bx,strategie
mov  al,cod
mov  ah,58H
int  21H
endm


FUNCTIA 59H
cit_eroare macro
mov  ah,59H
int  21H
endm


FUNCTIA 5AH
creez_temp macro pathnume,atribut
mov  cx,atribut
mov  dx,offset pathnume
mov  ah,5AH
int  21H
endm


FUNCTIA 5BH
creez_nou macro pathnume,atribut
mov  cx,atribut
mov  dx,offset pathnume
mov  ah,5BH
int  21H
endm
```

```
FUNCTIA 5C00H
LOCK macro handler,start,nr_octeti
mov  bx,handler
mov  cx,word ptr start
mov  dx,word ptr start+2
mov  si,word ptr nr_octeti
mov  di,word ptr nr_octeti+2
mov  al,0
mov  ah,5CH
int  21H
endm


FUNCTIA 5C01H
UNCLOCK macro
handler,start,nr_octeti
mov  bx,handler
mov  cx,word ptr start
mov  dx,word ptr start*2
mov  si,word ptr nr_octeti
mov  di,word ptr nr_octeti+2
mov  al,1
mov  ah,5CH
int  21H
endm


FUNCTIA 5E00H
cit_machine_nume macro bufer
mov  dx,offset bufer
mov  al,0
mov  ah,5EH
int  21H
endm


FUNCTIA 5E02H
printer_setup macro
index,lung,sir_secventa
mov  bx,index
mov  cx,lung
mov  dx,offset sir_secventa
mov  al,2
mov  ah,5EH
int  21H
endm


FUNCTIA 5F02H
cit_lista macro
index,local,distantat
```

```
mov  bx,index
mov  si,offset local
mov  di,offset distantat
mov  al,2
mov  ah,5FH
int  21H
endm
```

```
FUNCTIA 5F03H
REDIR macro
local,distantat,device,valoare
mov  bl,device
mov  cx,valoare
mov  si,offset local
mov  di,offset distantat
mov  al,3
mov  ah,5FH
int  21H
endm

FUNCTIA 5F04H
anulez_redir macro local
mov  si,offset local
mov  al,4
mov  ah,5FH
int  21H
endm

FUNCTIA 62H
get_PSP macro
mov  ah,62H
int  21H
endm
```

3. Interfata dintre limbajul Prolog si alte limbaje de programare

Turbo Prolog-ul permite incorporarea unor rutine scrise in limbajele C, Pascal, FORTRAN si asamblor.

Mai intii, predicatele incorporate in corpul unui program sursa Prolog si care sint scrise intr-unul din limbajele mentionate mai sus se vor declara printr-o specificare a limbajului sursa in care sint scrise fiind declarate ca predicate globale.

a) global predicates

scad(integer,integer,integer) - (i,i,o) language C

sau

b) global domains

lista1,lista2,lista3 = integer*

global predicates

adaug(lista1,lista2,lista3) - (i,i,o)(o,i,i) language Pascal

Astfel, in cazul a), predicatul scad are trei parametri intregi si este scris in limbajul sursa C primii doi parametri fiind valori de intrare (input) si cel de-al treilea fiind valoare rezultat de iesire (o - output).

In exemplul b) este nevoie de o declarare globala a tipului de lista de intregi al parametrilor predicatului adaug, scris in limbaj sursa Pascal, si mentionarea dublei actiuni a acestui predicat si anume primii doi parametrii sint de intrare definiti in sursa si cea de-a treia este valoarea de iesire si, la fel, predicatul adaug poate actiona cu ultimii doi parametrii ca valori de intrare si primul parametru ca valoare de iesire. Vom vedea, in continuare, ca specificarea variabilelor de intrare si iesire este esentiala in elaborarea acestor rutine.

Conventiile de apel ale subrutinelor si declararea parametrilor sint dictate de tipul microprocesoarelor cu care sint utilate calculatoarele, familia Intel permitind programatorului o alegere intre apelurile NEAR si cele FAR.

Turbo Prolog-ul pretinde ca toate apelurile si intoarcerile din ele sa fie de tipul FAR (codul sursa al rutinei si datele nu sint intr-un acelasi segment ci in zone distincte, arbitrare de memorie).

Interfata cu rutine scrise in limbajul C admite conventia ca parametrii sa fie depusi (push) in stiva in ordine inversa si, dupa terminarea rutinei, indicatorul de stiva (stack pointer) este ajustat automat. In celelalte limbaje parametrii sint dispusi in ordine normala in stiva, functia apelata fiind raspunzatoare de scoaterea parametrilor din stiva.

Un compilator de limbaj C folosit la compilarea de rutine utilizabile de Prolog trebuie sa satisfaca urmatoarele criterii:

a) posibilitateà de compilare pentru module de memorie largi;

b) compilatorul nu trebuie sa puna "-" dupa numele identificatorilor declarati public;

c) compilatorul trebuie sa fie capabil a produce fisierele direct compatibile cu link-editorul (link) sistemului DOS;

d) sa poata stopa procesul de verificare a stivei.

Dintre compilatoarele ce satisfac aceste prerogative mentionam: Turbo C, Microsoft C (vers. 4.*, 5.*), Lattice C, utilizabile cu optiunea "-v".

Pentru a putea accesa intreaga memorie interna, Turbo Prolog foloseste pointeri pe 32 de biti, orice parametru de iesire dintr-o rutina apelata fiind depus ca un pointer pe 32 de biti intr-o locatie unde valoarea de retur trebuie asignata.

Pentru parametrii de intrare valorile sint depuse direct iar marimea parametrului este cea a tipului sau declarat.

In Turbo Prolog tipurile datelor sint implementate astfel:

- Tip intreg (integer) - 2 octeti;
- Tip real (real) - 8 octeti (format (IEEE));
- Tip caracter (char) - 1 octet (ocupa 2 octeti cind este depus in stiva);
- Tip sir (string) - 4 octeti (ca pointer al unui sir cu terminator NULL (0));
- Tip simbol (symbol) - 4 octeti (ca pointer al unui sir cu terminator NULL (0));
- Tip compus (compound) - 4 octeti (ca pointer al unei inregistrari).

Am observat mai inainte ca un acelasi predicat declarat global in Prolog poate avea variante diferite functie de actiunea si tipul variabilelor sale parametrice. De aceea se impune conventia de denumire a fiecarei proceduri distincte pe care o determina formele multiple ale predicatului realizata prin familia de nume:

nume_predicat_X

cu X luind valori in multimea {0,1,2,...,N,...} ordonat crescator. Astfel, predicatul adaug, avind doua forme procedurale interne va fi denumit:

adaug_0 - pentru tipul (i,i,o)

si

adaug_1 - pentru tipul (o,i,i)

Aceste nume trebuiesc purtate de rutinele care definesc predicatul adaug, in limbajul sursa existind un izomorfism intre numerele adaugate rutinelor si formele parametrice ale variabilelor definite in definirea predicatului din cadrul sursei programului Prolog.

Sa exemplificam in continuare un program Prolog ce incorporeaza un predicat transf_lista ce are doua functii si este scris in limbaj C. Predicatul va avea doi parametri din care unul este o lista de intregi iar celalalt este o functie de argument intreg. Prima rutina trans_lista_0 transfera elementele listei de intrare ca argumente ale functiei de iesire f(x) = x-1 iar transf_lista_1 primeste ca argument valoarea functiei f(x) si o transforma in lista ale carei elemente vor fi pastrate in elementele listei initiale. Lista si functia le vom desemna prin numele

lista = integer*

functie = F(integer)

Ambele tipuri de variabile le vom reprezenta in limbajul C ca structuri avind grija de pastrarea conventiilor pentru cazurile cind sint variabile de intrare sau iesire.

Programul Prolog complet va da lista de intrare, va cere tiparirea valorilor functiei pentru fiecare element al listei dupa care va cere tiparirea listei patratelor elementelor initiale pornind de la valorile functiei ca parametrii de intrare.

```
global domains

lista = integer*

functie = F(integer)

global predicates

transf_lista (lista,functie) - (i,o)(o,i) language C
```

```
goal

transf_lista ([5],X) /*paseaza lui X valoarea X =
5-1 = 4*/

write (X), nl, transf_lista (Y,X) /*paseaza lui Y
valoarea Y = [(4+1)^2] = [25]*/

struct lista {

char tipl; /*tipul listei*/

int val; /*valoarea elementului curent in lista*/

struct lista *urmator; /*pointer pe urmatorul
element in lista*/

};

struct functie {

char tipf; /*tipul functiei*/

int val; /*valoarea argumentului*/

}

transf_lista_0 (intrare,iesire)

struct lista *intrare;

struct functie **iesire; /*se asigneaza ca pointer
pe 32 de biti*/

{

(*iesire) -> val = intrare -> val - 1 /*f(x) =
X-1*/

(*iesire) -> tipf = 1      /*tipul intreg al
argumentului*/

}

transf_lista_1 (iesire, intrare)

struct lista **iesire;

struct functie *intrare;

{

int i = 0;

i = intrare -> val; /*i = X-1 = 4*/

i = i+1; /*i = X+1 = 5*/

i = i*i; /*i = 5*5 = 25*/

(*iesire) -> val = 1;

(*iesire) -> tipl = 1 /*se specifica tipul listei*/

}
```

Un alt program ar putea fi cel ce calculeaza o functie trigonometrica in Prolog si anume sinusul unui unghi. Programul arata astfel:

```
global predicates

/*predicatul trig va da la iesire sinx*/
trig(real,real) - (i,o) language C

goal

write("Tastati un unghi"), readreal(X),

trig(X,Y),

write("Sinusul unghiului este",Y),nl.
```

Programul C care da valoarea lui trig este:

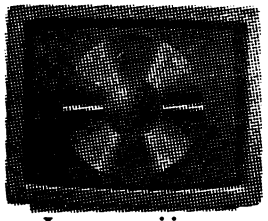
```
trig_0(intrare,iesire)

real intrare;

real * iesire;

{

*iesire = sin(intrare)

}
```

Pentru rularea si executia acestor programe se procedeaza astfel: se compileaza rutina C cu un compilator compatibil generind un modul obiect intr-un fisier, sa-i zicem C.OBJ. Se compileaza rutina Prolog cu compilatorul TURBO PROLOG generind un alt modul obiect, sa-i zicem P.OBJ. Se apeleaza apoi link-editorul sistemului DOS:

Link P.OBJ + C.OBJ,REZULTAT

ceea ce genereaza modulul executabil compus intr-un program pe care l-am numit REZULTAT ce este de tip .EXE. Apelul ulterior al acestui program executa cerintele celor doua module unite.

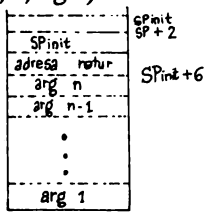
Cunoasterea modului in care se comporta stiva unui program in momentul apelarii unei rutine externe, ne va permite sa construim rutine scrise in limbaj de asamblare in care sa punem sau sa citim corect parametrii unei rutine apelate. Astfel, orice punct de intrerupere generat intr-un program printr-un apel de subrutina externa de tip CALL, sau prin numele direct al rutinei, salveaza pe doi octeti in stiva indicatorul anterior al stivei la momentul precedent apelului. Astfel, instructiunea PUSH BP (base pointer) salveaza in stiva indicatorul acesteia SP (stack pointer), marcind un nou SP la adresa SPnou = SP+2init.



Insa se mai impune cunoasterea locului in care a fost intrerupt programul initial pentru a ne intoarce in el imediat dupa ce s-a executat rutina apelata si, deci, se va salva in continuare in stiva adresa de retur in programul initial dupa actiunea subprogramului apelat. De obicei aceasta adresa este un cuvint (doi octeti) dar, in cazul nostru, cum apelurile sint de tip FAR, deci depasesc zona adresabila unui segment, aceasta adresa este de tip pointer ca patru octeti astfel ca indicatorul stivei se va actualiza la valoarea SPnou = SPinit+4.

Aceasta este starea stivei la momentul cind se preda controlul unei rutine externe. Daca aceasta are parametrii formali in componenta ei acestia se vor depune in stiva astfel:

nume_rutina(arg1,arg2,...,arg n)



ara Astfel, daca aceste argumente sint definite pe doi octeti avem: arg n se afla la adresa SPinit + 6, arg n-1 la adresa SPinit + 8 etc. In cazul apelurilor Prolog argumentele de intrare se salveaza pe doi octeti iar cele de iesire pe 4 octeti, astfel:

predicatul:

patrat(intrare,iesire) - (i,o) language asm

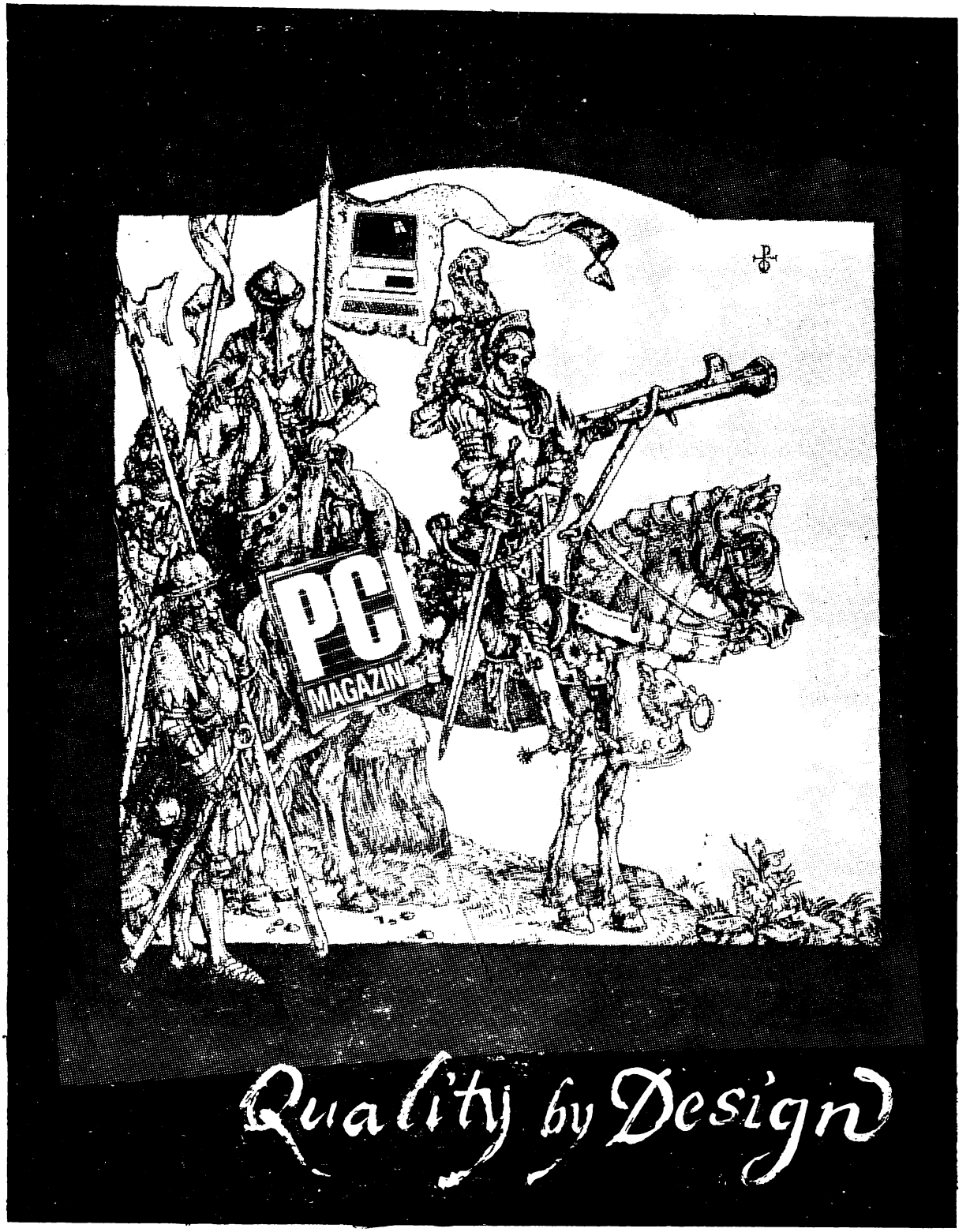
scris in limbaj de asamblare, care-si propune sa calculeze la iesire patratul valorii de intrare va avea valoarea parametrului de iesire salvat primul pe stiva la adresa SP + 6 (am presupus ca SP era indicatorul stivei anterior momentului de apel) iar parametrul de intrare se afla la adresa SP + 10, lungimea lui iesire fiind de patru octeti. Rutina in limbaj de asamblare s-ar scrie:

```
MOV  AX, [BP]*10      ;pune in AX valoarea de intrare

IMUL  AX, AX          ;calculeaza patratul valorii

LDS   SI, DWORD PTR[BP]+6 ;transfera in SI adresa valorii de iesire

MOV  [SI], AX         ;transfera la aceasta adresa patratul valorii de intrare
```

Programul Prolog se poate scrie:

```
global predicates

patrat(integer,integer) - (i,o) language asm

RUTINA  SEGMENT  BYTE

ASSUME  CS:RUTINA

PUBLIC patrat_0

patrat_0  PROC FAR

PUSH  BP

MOV  BP, SP

MOV  AX, [BP]+8

IMUL  AX, AX

LDS  SI, DWORD PTR [BP]+4

MOV  [SI], AX

POP  BP
```

```
MOV   SP, BP
RET   6
patrat_0    ENDP
RUTINA   ENDS
END
```



Despre arhitectura microcalculatoarelor IBM.

Extensiile optionale.

Adrian Popa

Succesul de care se bucura familia microcalculatoarelor compatibile IBM este astazi incontestabil. Insasi compatibilitatea la nivel utilizator pe care o ofera produsele diverselor firme trebuie privita ca un enorm avantaj. Astfel, putem beneficia de un hardware EPSON cu optiuni AST si software Microsoft, totul integrat standardului IBM.

Arhitectura nucleului acestor microcalculatoare a fost prezentata in numerele anterioare ale acestei reviste si va fi detaliata in numerele urmatoare. In principiu, utilizatorul poate interveni mai putin in aceasta parte, configuratia fiind standard si depinzind de posibilitatile financiare ale fiecaruia. Putem astfel avea un microprocesor 8086, 80186, 80286, 80386 sau 80486 cu avantajele de viteza si putere de calcul ale fiecarui tip, putem avea una sau doua unitati de floppy-disk de 5 ¼" sau 3 ½", putem avea sau nu un disc fix (Winchester), putem avea sau nu extensie de memorie. Modificarea sau extinderea configuratiei de baza este un exercitiu destul de dificil la nivel de amator si din cauza absentei componentelor VLSI specializate care realizeaza functiuni complexe la un pret de cost rezonabil. De exemplu un banc de memorie RAM de 1 MB cu timp de acces de 100 ns se prezinta sub forma unei placute cu conector JEDEC cu dimensiuni de numai 3x10 cm.

Indiferent de configuratia disponibila, sistemul de operare (de exemplu MS-DOS) descopera singur ce resurse are la dispozitie si face o configurare on-line pe hardware-ul existent. Alte date ce definesc configuratia sistemului se pastreaza intr-o memorie RAM nevolatila aflata de regula in chip-ul ceas/calendar (care are oricum o baterie sau acumulator tampon). Spre exemplu, discul fix poate avea intre 20 si 600 MB, numarul indicator al tipului efectiv folosit in sistem poate fi inscris in aceasta memorie de configurare (exista pina acum 47 de numere alocate diferitelor combinatii unitate/cuplor).

Sectiunea cea mai interesanta pentru utilizatorii de microcalculatoare compatibile IBM o reprezinta zona extensiilor optionale. Pentru conectarea optiunilor sint prevazuti un numar de conectori la care este adusa magistrala sistemului. Tipul conectorilor si alocarea semnalelor la pini sint standard, lucru care permite montarea oricarei optiuni, indiferent de firma producatoare, fara probleme de compatibilitate.

Soclurile pentru extensii permit conectarea la magistrala pe numai 8 linii de date (partea LOW) sau pe toate cele 16 linii de date. Folosirea integrala a magistralei de date la extensiile care permit acest lucru conduce la o crestere a vitezei de lucru dar exista microcalculatoare care nu sint prevazute cu conectorul pentru partea HIGH a magistralei.

Este deci vorba de doi conectori tip DDK 225D: unul de 60 de pini pentru partea LOW a magistralei de date si semnalele de comanda si un al doilea conector de 48 de pini pentru partea HIGH a magistralei de date si alte semnale suplimentare. Alocarea semnalelor la conectorul mare este prezentata in tabelul alaturat. Incarcarea pe fiecare linie si consumurile maximale pe sursele de 5V si +/-12V sint date de producatorul configuratiei de baza.

O latura a placii de extensie poate avea conectori pentru legarea diferitelor echipamente periferice, conectori ce trec prin peretele cutiei microcalculatorului si sint disponibili in exterior.

Numarul de optiuni disponibile astazi este foarte mare. Probabil ca in fiecare luna, undeva in lume un producator lanseaza pe piata o noua optiune. Standardul IBM aloca fiecarei optiuni o zona de adresa in spatiul I/O astfel ca si majoritatea acestor optiuni sint compatibile la nivel de programe utilizator.

Din lunga lista de optiuni amintim cuploarele video, modem-urile, procesoarele de comunicatie, procesoarele grafice, procesoarele de imagine video, cuploarele de retea NOVELL, cuploare de RAM- disk optic, cuploare de banda magnetica, etc.

Fiecare producator livreaza impreuna cu optiunea un manual de prezentare si o discheta cu

programele de activare a functiunilor extensiei. Programele se copiaza pe discul sistem si se lanseaza in executie la nevoie sau pot fi rezidente in memorie, caz in care ele sint activate prin AUTOEXEC.BAT, la lansarea sistemului.

Sa luam ca exemplu cuploarele video care ofera si multe alte facilitati in afara conectarii monitorului. Cuploarele video dezvoltate pina in prezent, care s-au impus ca standarde IBM, sint:

- MDA - Monochrome Display Adapter,
- HGC - Hercules Graphics Card,
- CGA - Color Graphics Adapter,
- EGA - Enhanced Graphics Adapter,
- VGA - Video Graphics Array.

Toate placile au un conector Cannon de 9 pini pentru cuplarea monitorului care poate fi alb/negru sau color. Placile MDA si HGC sint destinate monitoarelor alb/negru, celelalte se pot folosi atit pe monitoare color cit si pe monitoare alb/negru (in nuante de gri). De obicei o aceeasi placa ofera mai multe moduri de lucru.

Aceste placi ofera in plus si alte facilitati: un port serial si unul paralel (in acest caz sistemul de operare atribuie numele COM1, respectiv LPT1 acestor porturi), un cuplor game- joystick, ceas/calendar (in caz ca nu exista pe placa de baza), etc.

Dar despre facilitatile acestor extensii si despre alte optiuni in numerele viitoare ale revistei.

OPTION CARD CONNECTOR

PIN	NAME	DIRECTION	
1-2	GND	-	Ground
3-10	DTB0-7	I/O	Data bus low
11-12	-12V	-	Supply
13-20	ADR0-7	OUT	Address bus
21-28	-	-	NC
29-30	GND	-	Ground
31	CLK	OUT	System clock
32	GND	-	Ground
33	BSAK	OUT	Bus acknowledge
34	-	-	NC
35	IRD	OUT	I/O read control
36	IWR	OUT	I/O write control
37-38	-	-	NC
39	NRSIN	IN	Reset input
40	INT(H)1	IN	High priority interrupt 1
41	INT(H)2	IN	High priority interrupt 2
42	INT(L)	IN	Low priority interrupt
43	+5V	-	Supply
44	NRSET	OUT	Reset output
45-46	+5V	-	Supply
47	NDRQ(F)	IN	DMA request (master)
48	NDRQ(S)	IN	DMA request (slave)
49-51	-	-	NC
52	NIWS	OUT	I/O write control
53	NDAK(F)	OUT	DMA acknowledge (master)
54	NDAK(S)	OUT	DMA acknowledge (slave)
55	NEOP(F)	OUT	End of proc. (master)
56	NEOP(S)	OUT	End of proc. (slave)
57-58	+12V	-	Supply
59-60	GL	-	Ground

Din istoria calculatoarelor personale (II). Calculatoare personale in Romania.

Bodosi Imre

Calculatoarele personale au devenit o realitate pentru multi dintre noi. Mitul care le inconjura pina nu de mult a inceput sa se naruie. Copii se joaca cu ele, le deslusesc tainele acasa, in scoli, in tabere. Calculatorul personal este prezent pe biroul multor specialisti. Unii acum se imprietenesc cu el, ii incearca posibilitatile ca scula in cercetare si proiectare, altii au ajuns deja sa proiecteze asistati de calculator. Il regasim si in cele mai mici intreprinderi facind contabilitate, tinind gestiunea unor magazii, calculind retributii. Calculatoarele, datorita pretului relativ scazut s-au raspindit repede in cele mai diverse domenii, nivelul cererii fiind mereu peste cel al ofertei.

Microcalculatoarele romanesti au aparut in masa cu o intirziere de 5 pina la 10 ani fata de occident. Asamblarea lor in serie mare a fost posibila doar din momentul in care pretul componentelor a scazut la o valoare la care economia noastra si-a putut permite importarea lor sau cind industria autohtona a reusit sa asimileze aceste piese.

Cooperarea tehnico-stiintifica din cadrul CAER a contribuit la nasterea si la perpetuarea industriei de calculatoare, dar nu a putut determina o accelerare a dezvoltarii ei datorita unor mecanisme de conlucrare nestimulative. Diviziunea muncii, specializarea tarilor in producerea unor componente si subansamble, schimburi de rezultate tehnologice fie gratuite, fie decontate la nivelul cheltuielilor, au dus la diminuarea si, in unele cazuri, chiar la eliminarea concurentei de pe aceasta piata est-europeana. Fara ironie, putem afirma ca raminerea in urma a avut si avantaje. Ne-a dat o perspectiva asupra productiei mondiale de calculatoare, asupra tendintelor care au reusit sa se maturizeze. Astfel, am fost scutiti de a investi in directii inutile si a fost posibil ca familiile de microcalculatoare care s-au nascut cu intirziere sa urmeze niste linii stabile pe piata mondiala (Sinclair SPECTRUM, calculatoare CP/M, IBM PC). Astazi, a fi compatibil cu o familie de calculatoare din care exista milioane de exemplare in lume, inseamna, pe linga disponibilitatea unui software variat, ieftin si performant, si posibilitati vaste de cooperare. O astfel de familie nu poate avea o viata efemera, datorita multitudinii de interese concentrate in jurul ei si va cunoaste cu siguranta o dezvoltare de care vom beneficia si noi. Si nu vom pati ceea ce am patit cu FELIX-ul, si anume, am ramas singurii care-l dezvoltam, singurii care-l construim si singurii care-l utilizam.

Cererea de microcalculatoare pe piata interna s-a situat mereu la cote superioare posibilitatilor de productie. Aceasta cerere a stimulat nivelul si varietatea productiei autohtone, insa datorita, in primul rind, lipsei importului de calculatoare, a facut ca fabricantii interni sa prezinte calculatoare fabricate cu tehnologii ramase in urma, nefiabile, la preturi cu mult peste valoarea lor de utilizare.

Calculatorul familial (Home computer)

Primele calculatoare familiale s-au proiectat la Institutul Politehnic Bucuresti (aMIC, HC-80) si la Institutul pentru Tehnica de Calcul, filiala Cluj (PRAE) in 1983 si au intrat in fabricatie de serie in 1985, dupa opt ani de la primul calculator personal, Apple II al lui Steve Jobs si Stephen Wozniak. Cu toate ca din punct de vedere hardware sint niste calculatoare reusite, performante in categoria lor, nu au cunoscut o raspindire in masa datorita saraciei in programe: compilatoare, aplicatii generice, jocuri. Fiind constructii originale nu se pot rula pe ele decit programe anume scrise. Dar imediat dupa aceste cautari de drum au aparut calculatoarele compatibile SPECTRUM, care aveau deja asigurate biblioteci intregi de programe scrise pentru Sinclair SPECTRUM. TIM-S, produs de Fabrica de Memorii Timisoara, si HC-85, produs de Fabrica de Calculatoare Electronice, au ajuns repede pe mina copiilor, acestia fiind atrasi mai ales

de grafica, coloristica bogata si sonoritatea mirifica a jocurilor.

TIM-S si HC-85 sint construite in jurul microprocesorului Z80 si au o memorie interna de 48 pina la 64 Kocteti. Ele sint comercializate sub forma unei tastaturi care inglobeaza si calculatorul propriu-zis, la care se leaga televizorul pentru afisare si casetofonul audio pentru inregistrarea si redarea programelor si datelor.

- gestiuni mici, de exemplu in mediul familial;
- programe educative (simulari animate de reactii chimice, flux de curent in circuite electrice, diverse scheme etc).

Calculatoare semiprofesionale (CP/M)

Este categoria din care s-au produs cea mai mare varietate si cele mai largi serii. Domeniul cucerit este cel al problemelor financiar-contabile, de gestiune,

Tipul	M118B	CUB-Z	TPD	JUNIOR
Intreprinderea producatoare	ICE	ICE	IEPER	IEPER
Anul intrarii in fabricatie	1981	1984	1986	1985
Microprocesor	8080	Z80	8080/Z80	Z80
Optiune grafica	da	da	da	-
Disc flexibil	2x8"	2x8" 2x5 1/4"	2x8"	2x8" 2x5 1/4"
Interfata paralela	da	da	da	-
Interfata seriala	da	da	da	da
Pret orientativ fara imprimanta	300.000	170.000	230.000	150.000

Figura 1. Calculatoare CP/M

Aceste microcalculatoare nu au un sistem de operare propriu-zis, "se trezesc" in Basic deorece acest interpretor este rezident in memoria lor interna. Pe casete sint insa disponibile si compilatoare pentru alte limbaje: Pascal, Forth, microPROLOG, LOGO, precum si nenumarate jocuri, editoare de texte, biblioteci de rutine stiintifice. Microcalculatoarele nu sint doar simpli parteneri de joc ci, datorita posibilitatilor de programare, pot executa:

- programe de calcul stiintific (cu limitari la dimensiunile tablourilor si la viteza de calcul);
- invatare programata;

in intreprinderi mici. In unele locuri au patruns dupa masinile de facturat si contabilizat pe care le-au inlocuit dar, in majoritatea cazurilor, au gasit teren virgin, fiind primii reprezentanti ai erei calculatoarelor.

Aceste calculatoare au o serie de trasaturi comune:

- sint microcalculatoare de 8 biti, cu microprocesoare Intel 8080 sau Z80;
- au un sistem de operare compatibil CP/M, incarcabil de pe disc flexibil;

- memoria interna se rezuma la cel mult 64 Kocteti, microprocesoarele 8080 si Z80 neputind adresa mai mult;
- ca memorie externa au, de regula, doua unitati de discuri flexibile (8" sau 5 ¼");
- afisarea se face pe monitor alb-negru, de regula cu 24 linii si 80 de coloane;
- poseda o interfata seriala (V24, RS232) care permite conectarea lor la o retea de telecomunicatii.

Diferentele intre familii sint la viteza de executie, numarul si tipurile de interfete si, nu in ultima instanta, in fiabiliatea si posibilitatile de service. Cele mai raspindite modele de calculatoare din aceasta clasa sint ilustrate, comparativ, in figura 1.

Alte calculatoare din aceasta categorie, produse in serii mai mici sau in conditii de microproductie, sint:

- MS-100 (ICSIT-TCI Filiala Timisoara si Fabrica de Memorii Timisoara);
- CE-119S (ICSIT-TCI);
- COBRA (ICSIT-TCI Filiala Brasov);
- TELEROM P386 (IEIA Cluj-Napoca).

Compatibilitatea cu sistemul de operare CP/M a acestor calculatoare asigura posibilitatea utilizarii a sute de programe: compilatoare, aplicatii generice, utilitare.

Compilatoarele cele mai reprezentative, disponibile sub CP/M, sint: Basic, GW-Basic, FORTRAN, COBOL, C, Turbo Pascal, Pascal MT+.

Editarea de documente. Cel mai raspindit editor de texte sub CP/M este WordStar, editor cu care se pot crea documente la nivelul unei imprimante matriciale cu care se comercializeaza de regula calculatoarele CP/M. O problema doar partial rezolvata este editarea literelor specifice alfabetului romanesc. Imprimantele cunosc acest alfabet, dar nu si tastaturile si generatorul de caractere pentru ecran. Astfel, pentru a imprima un text in romaneste esti nevoit sa tastezi acolade, paranteze drepte in locul literelor specifice, romanesti, aceste acolade si paranteze aparind si in textul de pe ecran, pentru ca in final pe imprimanta sa devina literele asteptate: a, i sau t.

Grafica. Nu exista un standard de grafica pentru calculatoarele CP/M si nici nu exista optiune grafica la toate modelele. Din acest motiv pachetele si aplicatiile grafice nu sint raspindite.

Gestiunea bazelor de date. dBASE II, in 64 de Kocteti, poate realiza toate functiile unui sistem de gestiune a bazelor de date (SGBD) veritabil: definirea structurii bazei de date, introducere de date, actualizare, interogare, generare de rapoarte. Pacat ca numai memorii externe de cel mult 600 Kocteti ii stau la dispozitie. Cu dBASE se genereaza foarte usor aplicatii din domeniul financiar-contabil, personal-retribuire, gestiune, si aceste aplicatii se intretin foarte usor. Aceste aplicatii, insa, nu se pot integra intr-un sistem informatic pe calculatoare CP/M datorita spatiului restrins de memorie externa. O integrare ulterioara a lor este posibila doar sub dBASE III/IV pe calculatoare compatibile IBM XT avind disc de masa.

In dBASE se pot scrie aplicatii "prietenoase" care se opereaza pe baza de "meniuri" si care nu se "supara" cind primesc comenzi pe care nu le inteleg, ci il ajuta pe utilizator la tot pasul prin mesaje explicative si liste de comenzi permise.

Comunicatii. Prin interfata seriala calculatoarele CP/M pot fi conectate la o retea de telecomunicatii. Pachetul de programe cel mai des utilizat in comunicatii este Kermit, cu care se pot transfera fisiere la distanta catre si dinspre cele mai diverse tipuri de calculatoare.

In anii '80 calculatoarele CP/M s-au raspindit in masa datorita disponibilitatii practic exclusive pe piata interna, si aceasta in ciuda unor deficiente, cum ar fi:

- capacitate si viteza de prelucrare mica pentru a rezolva integral gestiunea economica, chiar intr-o intreprindere mica;
- fiabilitate redusa;
- pretul ridicat, comparativ cu modelele similare din strainatate si chiar comparativ cu modele de capacitate mai mare (IBM PC) provenite din import-est si vindute in tara in loturi mici.

Calculatoare personale profesionale (compatibile IBM PC)

Daca microcalculatoarele semiprofesionale au invadat intreprinderile si institutiile, nu acelasi lucru il putem afirma despre cele compatibile IBM PC. Acestea s-au produs in serii mici datorita, lipsei de piese si subansamble. Cu toate ca, de exemplu, primul FELIX PC s-a nascut in 1985 si a intrat in

fabricatie de serie in 1986, nici acum, dupa 4-5 ani, nu a ajuns sa aiba o familie numeroasa.

Alte initiative in domeniul calculatoarelor compatibile IBM PC:

- JUNIOR XT, intrat in fabricatie in 1988 la IEPER;
- TELEROM PC fabricat de IEIA Cluj-Napoca.

Si importul de calculatoare IBM PC s-a situat la nivel scazut. Citeva loturi mai insemnate s-au importat de la firma ROBOTRON si din Iugoslavia. O lista cu cele mai cunoscute tipuri este prezentata in figura 2.

care le asigura PC-ul: viteza de excutie mare, grafica, memorie externa de masa etc. Editarea de documente, datorita imprimantelor cu laser (Hewlett- Packard LaserJet, XEROX 4045) si unor pachete complexe de redactare a documentelor (Word Perfect, XEROX Ventura Publisher) ajunge pina la nivelul la care se pot redacta pagini gata de tipar (Desktop Publishing). Nivelul graficii pe calculator a crescut foarte mult. Datorita standardelor grafice disponibile, pe IBM PC s-au putut dezvolta niste pachete grafice sofisticate (cum ar fi AutoCAD, al firmei AutoDesk). Rezolutia din ce in ce mai mare a monitoarelor si a imprimantelor cu laser fac sa creasca in aceeasi masura calitatea desenelor executate pe calculator.

	MicroBIT XT	MicroBIT AT	ROBOTRON EC1834
Microprocesor	8088	8086	8086
Memorie interna	640 Ko	1 Mo	640 Ko
Monitor	color grafic	color grafic	monocrom alfanum.
Disc Winchester	20 Mb	20 Mb	40 Mb
Disc flexibil	1x360 Ko	1x1.2 Ko	2x720 Ko
Interfata paralela	da	da	da
Interfata seriala	da	da	-
Pret (lei)	66.868	61.500	212.388 fara impr.

Figura 2. Calculatoare compatibile IBM PC

Datorita instalarilor recente nu putem vorbi deocamdata de o istorie a utilizarii calculatoarelor IBM PC la nivel national, asa ca din acest moment trebuie sa "comutam" la timpul viitor si vom vorbi, deci, despre perspectivele lor.

Lucrarile proiectate pentru calculatoare CP/M si executate pe acestea pot fi trecute cu usurinta pe IBM PC. Instalate pe acestea din urma, lor li se deschid noi perspective datorita posibilitatilor pe

Calculatoarele personale de tip IBM PC vor fi cu siguranta calculatoarele viitorului in intreprinderi mici si mijlocii. Discul de masa si viteza de executie mare fac posibila implementarea unor aplicatii integrate de gestiune cu dBASE III/IV, FOXBASE, CLIPPER sau chiar organizarea unor baze de date distribuite (SQL). Retelele locale cu servicile lor variate pot rezolva schimbul de informatii dintre sectoarele de activitate din cadrul unei unitati economice.

Limbaje de programare in design-ul sistemelor de operare si al aplicatiilor.

Constantin Mihai

Limbajul C (III)
Declaratii, Operatori, Conversii de tip

In acest articol vom prezenta declaratiile si tipurile de date fundamentale manipulate in limbajul C. Pe parcursul prezentarii, unele din lucrurile expuse in primele doua articole vor fi reluate si explicitate. In plus, va fi explicata folosirea operatorilor aritmetici, insistind asupra conceptelor de precedenta si asociativitate. In expresiile cu operanzi de tipuri diferite, apar anumite conversii implicite. Sint explicate regulile de conversie, incluzind fortarea (cast) conversiei explicite.

1. Declaratii

Variabilele si constantele sint obiectele manipulate de catre un program. In general, toate variabilele trebuiesc declarate inainte de a fi folosite. De exemplu:

```
int i, j, k;
float lungime, latime;
char c;
```

sint declaratii care specifica faptul ca i, j si k sint variabile de tipul `int`, ca lungime si latime sint variabile de tipul `float` si ca c este o variabila de tip `char`. Un nume de variabila este un identificator si o declaratie va fi formata dintr-un tip urmat de o lista de nume de variabile terminate cu ";".

Inceputul unui program poate arata in felul urmator:

```
main () {
     /* declaratii */
     int a, b;
     char c1, c2;
     float numar_mare;
     /* instructiuni */
     . . .
}
```

Acoladele { si } delimiteaza un bloc . Un asemenea bloc mai poarta denumirea de instructiune compusa si poate contine declaratii si instructiuni. Declaratiile, daca exista, trebuie sa apara inaintea instructiunilor. Declaratiile servesc doua scopuri principale. Mai intii, indica compilatorului sa rezerve un anumit spatiu de memorie pentru a retine valorile asociate variabilelor; in al doilea rind, declaratiile specifica tipul datelor asociate variabilelor. In implementarile recente, declaratiile se folosesc si in scopul initializarii variabilelor. Să consideram codul urmator:

```
     int a, b, c;
     float x, y, z;

     a = 2;
     b = -3;
/* adunarea variabilelor int */
     c = a+b;

     x = 3.2;
     y = -7.7;
/* adunarea variabilelor float */
     z = x+y;
```

In instructiunea `c = a+b;` operatia + este aplicata unor variabile de tip `int`, ceea ce la nivelul masinii se reflecta diferit de operatia + aplicata variabilelor de tip `float`. Desigur ca acesta este un aspect transparent programatorului de C, dar compilatorul trebuie sa recunoasca diferenta si sa genereze instructiunile masina potrivite. Iata citeva exemple in care declaratiile sint folosite si pentru initializarea variabilelor:

```
char backslash = '\\';
int i = 0;
float eps = 1.0e-5;
```

2. Variabile externe

Variabilele declarate intr-un bloc sint asociate acestuia si in concluzie nici o alta functie nu poate avea acces direct la ele. Fiecare variabila locala unei

rutine prinde viata doar in momentul in care functia este apelata, si dispare atunci cind parasim functia. Acesta este motivul pentru care asemenea variabile se numesc **automate**. Vom continua sa folosim termenul "automat" pentru a referi aceste variabile dinamice locale.

Deoarece variabilele automate "vin si se duc" odata cu invocarea functiei, ele nu-si retin valorile intre doua apelari succesive, si trebuiesc setate explicit la fiecare intrare in rutina. Daca ele nu sint setate valorile corespunzatoare vor fi nedefinite (`garbage`). Intr-un articol viitor vom trata despre o alta clasa de variabile, **statice**, ce-si retin valoarea intre doua apelari.

Ca o alternativa la variabilele automate, este posibil sa definim variabile **externe** tuturor functiilor, variabile globale care pot fi accesate dupa nume de catre orice functie. (Acest mecanism este asemanator instructiunilor COMMON din Fortran sau EXTERNAL din PL/I). Deoarece variabilele externe sint global accesibile, ele pot fi folosite in locul listelor de argumente pentru a comunica date intre functii. Mai mult, deoarece variabilele externe ramin in viata permanent, fara sa dispara la iesirea din functie, ele pot sa-si retina valorile si dupa terminarea executiei rutinelor care le-au setat aceste valori.

O variabila externa va trebui sa fie definita in afara oricarei functii; aceasta va aloca spatiul necesar. De asemenea, variabila trebuie sa fie "declarata" in fiecare functie care doreste accesul la ea; aceasta se va realiza fie explicit printr-o declaratie `extern` sau implicit din context. Concretizam discutia noastra cu programul urmator, care, desi contine notiuni inca necunoscute dumneavoastra, constituie un exemplu coerent de sursa C.

Variabilele externe din `main()` si `getline()` sint "definite" prin primele linii ale exemplului de mai sus, linii care le stabilesc tipul si determina alocarea de spatiu pentru ele. Inainte de a folosi intr-o functie o variabila externa, numele acesteia

```
#define MAXLINE 100     /* lungimea maxima a unei linii */

char line   [MAXLINE]; /* linia de intrare */
char save   [MAXLINE]; /* vector folosit pentru retinerea celei
                          mai lungi linii */
int max; /* marimea celei mai lungi linii */

main () {
/*
   programul preia de la tastatura linii de text,
   retinind-o pe cea mai lunga.
*/
     int len;
     extern int max;
     extern char save [];

     max = 0;
     while ((len = getline ()) > 0)
          if (len > max) {
               max = len;
               strncpy (buffer, line, len);
          }
          if (max > 0)     /* daca a existat cel putin o linie */
               printf ("%s", save);
}

getline () {
     int c, i;
```

```
    extern char line [];

    for (i = 0; i < MAXLINE-1  && (c=getchar()) != EOF && c != '\n'; ++i)
         line [i] = c;
    if (c == '\n') {
         line [i] = c;
         ++i;
    }
    line [i] = '\0';
    return (i);
}
```

trebuie facut cunoscut functiei. Un mod de a realiza acest lucru este de a scrie o "declaratie" externa. In cazul in care definitia variabilei apare, in fisierul sursa, inaintea unei functii particulare, nu mai este nevoie de declaratia `extern`. Din acest motiv declaratiile externe facute in functiile `main()` si `getline()` sint redundante.

Din cele de mai sus s-a putut desprinde distinctia intre "declaratie" si "definitie". "Definitia" apare in locul in care variabila este creata si ii este asignat spatiul necesar; "declaratia" apare in locurile in care este stabilita natura variabilei fara a i se aloca spatiu.

Exista tendinta de a trata cit mai multe variabile ca externe, deoarece aceasta pare a simplifica oarecum comunicatia. Acest stil de programare conduce la modificari neasteptate si nedorite ale variabilelor globale, logica programului fiind din ce in mai greu de controlat.

Variabilele externe si cele statice sint initializate cu zero in mod implicit.

3. Operatori aritmetici

Operatorii aritmetici binari sint +, -, *, / si operatorul modul %. Mai exista un minus unar -, dar nu exista + unar.

Impartirea intreaga truncheaza partea fractionara. Expresia

```
x % y
```

produce restul impartirii lui x la y, si va fi zero atunci cind x se divide cu y. De exemplu, un an bisect este definit ca fiind divizibil cu 4 dar nu cu 100, exceptind anii divizibili cu 400. De aceea:

```
if (an % 4 == 0 &&
     an % 100 != 0 ||
     an % 400 == 0)
     "anul este bisect"
else
     "anul nu este bisect"
```

Operatorul % nu poate fi aplicat tipurilor `float` si `double`. Operatorii + si - au aceeasi precedenta, care este mai scazuta decit precedenta operatorilor *, / si %, care la rindul lor au o prioritate mai scazuta decit minusul unar. Ordinea evaluarii nu este specificata pentru operatori asociativi si comutativi ca + si *; compilatorul poate rearanja parantezarea calculelor. De aceea expresia `a+(b+c)` poate fi evaluata sub forma `(a+b)+c`.

4. Operatorii logici si relationali.

Operatorii relationali sint urmatorii:

```
<, <=, >=, >
```

Toti au aceeasi precedenta. O precedenta imediat inferioara o au operatorii de egalitate:

```
==, !=
```

Operatorii relationali au o precedenta inferioara operatorilor aritmetici, deci o expresie de tipul i < a-1 va fi evaluata ca i < (a-1). Expresiile logice se pot conecta prin intermediul conectorilor logici && si ||. Expresiile conectate prin && si || sint evaluate de la stinga la dreapta. Precedenta lui && este mai mare decit cea a lui ||, si ambele sint mai mici decit cea a operatorilor relationali si a celor de egalitate, astfel ca expresii ca

```
i < a-1 && (c = getchar ()) != '\n'
&& c != EOF
```

nu necesita paranteze suplimentare. Dar deoarece precedenta lui != este mai mare decit cea a atribuirii, sint necesare parantezele:

```
(c = getchar ()) != '\n'
```

pentru a obtine rezultatul dorit.

Operatorul de negatie unar ! converteste un operand nenul sau adevarat in zero, si un operand zero sau fals in 1. Operatorul ! mai apare in constructii de genul:

```
if (! flag)
```

in loc de

```
if (flag == 0)
```

5. Conversii de tip

Atunci cind in expresii apar operanzi de tipuri diferite, acestea sint convertite la un tip comun pe baza unui numar redus de reguli.

Mai intii, variabilele de tip `char` si `int` se pot combina in mod liber in expresiile aritmetice; orice `char` intr-o expresie este automat convertit la un `int`. Exemplificam aceasta regula prin functia `atoi()`, care converteste un sir de cifre in echivalentul sau numeric.

```
atoi (char s []) {
     int i, n;

     n = 0;
     for (i=0; s [i] >= '0' &&
          s [i] <= '9'; ++i)
          n = 10 * n + s [i] - '0';
     return (n);
}
```

La convertirea unui `char` in `int` apare urmatoarea problema: limbajul nu verifica daca variabilele de tip `char` sint cantitati cu semn sau fara semn. In cazul unei asemenea conversii poate apare foarte usor un intreg negativ. Din nefericire nu putem da o regula pentru aceasta, fenomenul fiind dependent de masina.

O alta forma utila de conversie automata este cea care atribuie expresiilor logice valoarea intreaga 1 pentru adevarat si 0 pentru fals.

In general, in cazul operatorilor binari cum ar fi + sau * rezultatul va avea tipul "cel mai complex" dintre tipurile celor doi operanzi. Mai precis, pentru fiecare operator aritmetic, urmatoarea secventa de reguli de conversie este aplicata:

- `char` si `short sint` convertite la `int`, si `float` este convertit la `double`.
- daca unul dintre operanzi are tipul `double`, celalalt va fi convertit la `double`, iar rezultatul va fi `double`.
- daca unul din operanzi are tipul `long`, celalalt este convertit la `long`, iar rezultatul va fi `long`.
- daca unul din operanzi are tipul `unsigned`, celalalt va fi convertit la `unsigned si` rezultatul va fi `unsigned`.
- altfel operanzii au tipul `int`, iar rezultatul va fi `int`.

De notat ca toata aritmetica flotanta din C este facuta in dubla precizie. Toti operanzii flotanti sint automat convertiti la `double`.

In cazul atribuirilor, valoarea din dreapta este convertita la tipul din stinga, care este tipul rezultatului.

In cazul in care argumentul unei functii este o expresie, conversiile de tip au deasemenea loc la transmiterea argumentelor unei functii: in particular `char` si `short` devin `int`, iar `float` devine `double`.

In fine, putem forta tipul oricarei expresii cu o constructie denumita "cast".

In constructia `(tip) expresie`, expresia este convertita la tipul specificat. De exemplu, rutina de biblioteca `sqrt()` asteapta un argument `double`. Deci, daca `n` este un intreg, in expresia `sqrt ((double) n)`, acesta va fi mai intii convertit la `double`. Regula de fortare a tipului (`casting`) are aceeasi precedenta cu cea a operatorilor unari.

Bibliografie:

- *The C Programming Language*, Kernighan Brian, Ritchie Dennis;
- *A Book on C*, Kelley Al, Pohl Ira, 1986;
- *Standard C*, Plauger P.J., Brodie Jim, 1989.

Initiere in conceptele comunicatiei de date (II)

Octavian Paiu

1. Modem-uri

Modem-urile sint dispozitive care realizeaza transmisia de date numerice pe canale (linii) analogice. La emisie, modem-urile moduleaza semnalele numerice (impulsuri) in semnale analogice (de regula in benzi audio), iar la receptie demoduleaza semnalele analogice din canalul de comunicatie in semnale numerice (vezi figura 1).

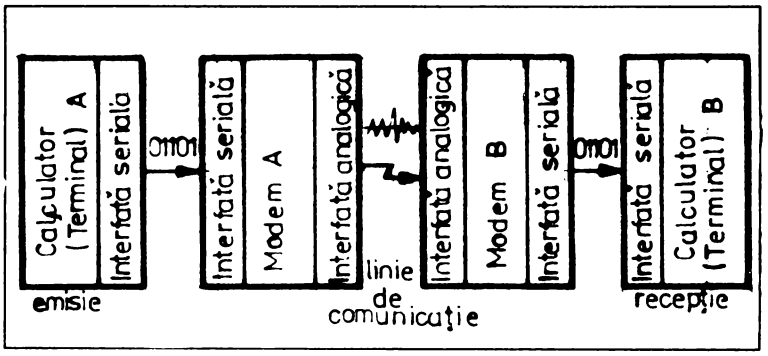


Figura 1. Conectarea a doua calculatoare utilizind modem-uri

Modem-urile (denumite generic DCE-Data Comunications Equipment) se conecteaza pe de o parte la calculator (prin interfata seriala), iar pe de alta parte la canalul de comunicatie, constituind astfel "intermediari" intre calculator (generic DTE-Data Terminal Equipment) si canalul (linia) de comunicatie.

De regula, modem-urile pot fi semi-duplex sau duplex, dar exista si modem-uri care pot lucra in ambele moduri. In general modem-urile semi-duplex se utilizeaza pentru transmisii sincrone de date.

1.1. Semnalul purtatoare

Purtatoarea este un semnal electric analogic de o anumita frecventa. Acest semnal este modulat de catre semnalul numeric (impulsuri) folosind diverse tehnici (modulatie de amplitudine, modulatie de faza etc.).

Modem-urile semi-duplex utilizeaza o singura frecventa pentru purtatoare, iar cele duplex dispun de doua frecvente distincte, pentru fiecare directie de transmisie in parte.

Modem-ul apelat trebuie sa-si plaseze primul purtatoarea (de raspuns) in canalul de comunicatie. Dupa detectia purtatoarei de raspuns de catre modem-ul apelator acesta plaseaza cealalta purtatoare (originara) in canal. In acest fel, dupa ce ambele frecvente purtatoare sint prezente in linia de comunicatie, se realizeaza conectarea "logica" a modem-urilor si se poate incepe traficul de date.

1.2. Modem-uri asincrone

De regula, aceste modem-uri sint duplex si functioneaza la viteze de la 1200 la 9600 bps (uneori la viteze si mai mari). Pentru fiecare din aceste viteze de transmisie s-au stabilit standarde electrice de interfata (analogica si seriala numerica) care sint respectate in general de catre toti producatorii de modem-uri. Cu cit viteza de functionare a unui modem este mai mare cu atit si calitatea liniei de comunicatie trebuie sa fie mai buna.

Standardul 212

Modem-urile construite dupa acest standard lucreaza la viteza de 1200 bps. Unele modem-uri de acest tip pot insa functiona si la viteza de 300 bps (viteza de "repliere"-in cazul in care calitatea liniei nu este satisfacatoare).

Standardul 224

Aceste modem-uri functioneaza, in mod "normal", la viteza de 2400 bps. Standardul american 224 corespunde cu standardul european CCITT V.22 bis. De asemenea, si aceste modem-uri pot functiona cu o viteza de "repliere" de 1200 bps.

Modem-uri rapide

Sint considerate rapide acele modem-uri care functioneaza la viteze de 9600 bps sau chiar mai mari (de exemplu 19200 bps). Modem-urile rapide nu sint standardizate inca total. Exista standardul european CCITT V.32 care defineste modem-uri rapide (9600 bps) duplex, sincrone sau asincrone, care pot functiona pe linii telefonice comutate sau inchiriate. Dar multi producatori de astfel de modem-uri nu respecta integral recomandarile V.32 mentinind doar o compatibilitate minimala cu acest standard. De regula, modem-urile de 19200 bps nu sint compatibile intre ele, fiecare functionind conform specificatiilor proprii ale producatorului.

Moduri de cuplare la canalul de comunicatie

In functie de tipul modem-ului exista doua moduri de a-l cupla la canalul de comunicatie: cuplare acustica si cuplare "directa" (electrica). Modem-urile cu cuplare acustica sint modem-uri ieftine si cu performante scazute.

Cele mai utilizate modem-uri sint cele cu cuplaj direct (electric) la linia de comunicatie. Chiar si pentru acest tip de modem, performantele depind in buna masura de calitatea liniei (canalului) de comunicatie si de viteza de transmisie utilizata.

1.3. Controlul modem-urilor

Gestiunea unei conexiuni "logice" intre doua modem-uri implica, de regula, urmatoarele actiuni principale:

- apelul;
- raspunsul;
- transmisia de date;
- deconectarea "logica".

Modem-urile cu cuplare directa obisnuite au posibilitatea de a "raspunde" automat la apelarea numarului telefonic la care sint conectate. De asemenea, pot realiza deconectarea automata. In plus, modem-urile cu cuplare directa "inteligente" pot realiza apelarea automata a unui numar de telefon (al modem-ului corespondent).

Controlul prin interfata seriala RS-232

Modem-urile obisnuite (dar si cele inteligente) pot fi controlate numai prin interfata RS-232, utilizindu-se semnalele DTR si DCD.

Semnalul de la pozitia (pinul) 20 (DTR-Data Terminal Ready) se foloseste de catre terminal (sau calculatorul cu interfata seriala DTE) pentru a indica modem-ului starea de "pregatit" pentru prelucrare de date. Daca acest semnal are valoarea logica "0" se "forteaza" modem-ul sa intrerupa conexiunea logica.

Semnalul de la pozitia (pinul) 8 (DCD-Data Carrier Detect, detectie purtatoare) se foloseste de catre modem pentru a indica terminalului (sau calculatorului cu interfata de tip DTE) realizarea conexiunii logice cu modem-ul (si terminalul) corespondent.

Deseori modem-urile sint livrate de la producator configurate astfel incit sa ignore semnalele de control. In manualele tehnice ale diverselor tipuri de modem-uri sint indicate procedurile de pozitionare a unor comutatoare sau "jumpere" pentru activarea acestor semnale.

Controlul modem-urilor inteligente

In afara posibilitatii de control prin semnale electrice de interfata, modem-urile cu cuplare directa inteligente pot fi controlate si prin caractere de control transmise de catre terminal/calculator. Pentru a interpreta caracterele drept caractere de control, modem-ul trebuie sa fie trecut in "modul comanda" (alternativa la "modul date" in care caracterele sint vehiculate de catre modem intre calculator si canalul de comunicatie).

La aplicarea tensiunii de alimentare acest tip de modem-uri trece in mod automat in modul comanda. Comenzile transmise modem-ului pot fi folosite pentru: apel, raspuns sau, in unele situatii, intreruperea conectarii "logice".

De exemplu, comanda de apel realizeaza apelul automat al corespondentului. Dupa stabilirea conexiunii logice, modem-ul trece automat in modul date.

Trecerea in modul comanda (din modul date) se poate face oricind prin emiterea de catre calculator (DTE) a unei secvente speciale de caractere de control (secvente ESCAPE).

De exemplu, pentru modem-urile compatibile Hayes, secventa de ESCAPE este urmatoarea: o pauza de o secunda urmata de trei caractere " + " si din nou o pauza de o secunda.

Unele modem-uri inteligente permit folosirea modului comanda impreuna cu semnalele de interfata RS-232 sau permit controlul numai prin utilizarea semnalelor RS-232.

Transparenta modem-urilor inteligente

Modem-urile care trec din modul date in modul comanda la sesizarea unei secvente de caractere nu sint "transparente" total la datele vehiculate. Aceasta lipsa poate cauza probleme la transmisiile de date binare care pot contine orice secventa de caractere.

binare (de exemplu secventa descrisa mai sus utilizata de modem-urile compatibile Hayes).

Functionarea cu raspuns automat

Multe tipuri de modem-uri cu cuplaj direct la linia de comunicatie, atit obisnuite cit si inteligente, pot fi configurate (prin hardware) astfel incit sa "raspunda" automat la realizarea purtatoarei de la corespondent si sa intrerupa legatura logica la disparitia purtatoarei. Aceasta facilitate permite realizarea usoara a unei cooperari intre modem si software-ul de comunicatie din calculator.

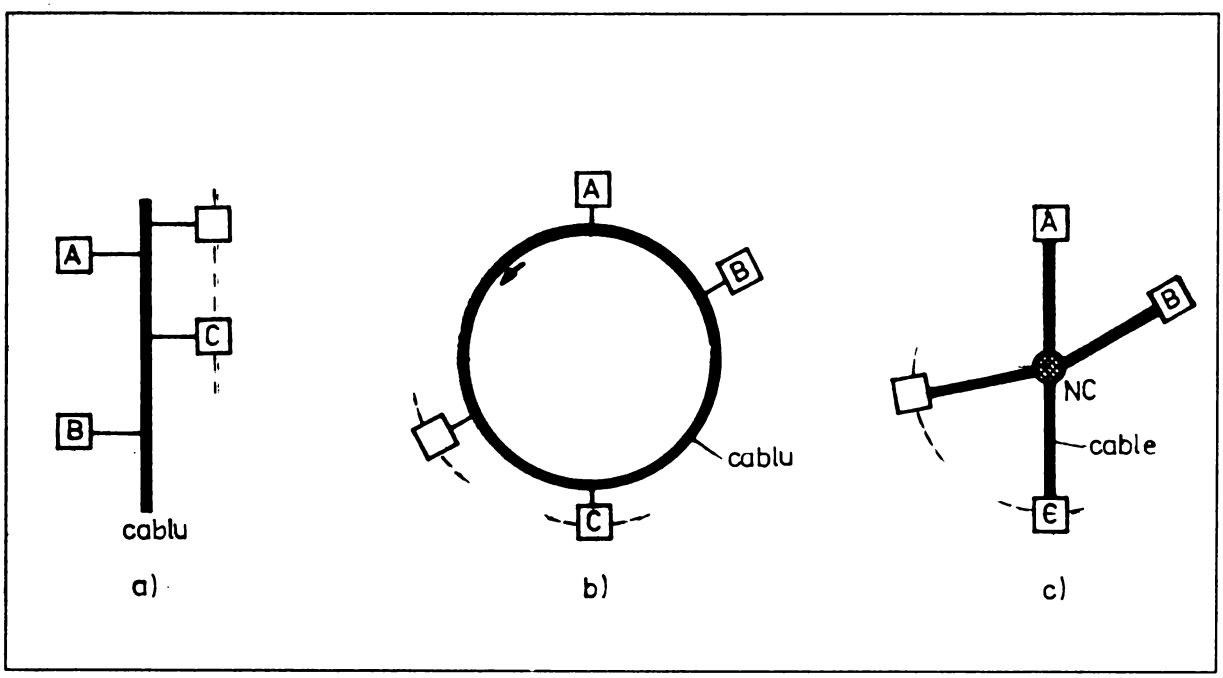


Figura 2. Topologii uzuale pentru LAN

Producatorii de modem-uri inteligente rezolva problema transparentei, de exemplu, in urmatoarele moduri:

- Nerecunoasterea secventelor speciale de caractere ESCAPE. Trecerea in modul comanda se poate face numai prin: intreruperea legaturii de catre modem-ul corespondent, dezactivarea de catre calculator a semnalului de interfata DTR sau transmiterea catre modem a unui semnal BREAK.
- Folosirea unor secvente de trecere (din modul date in modul comanda) care sa nu poata fi generate accidental prin transmiterea unor date

2. Retele pentru comunicatia de date

Retelele permit conectarea intre ele a diferitelor "puncte" in care se prelucreaza date. Retelele (atit generale cit si locale) contin o multitudine de puncte de acces la datele vehiculate in retea.

2.1. Circuite fizice si circuite virtuale

Circuitele care se realizeaza numai prin conectarea fizica (cu linii fizice) a doua sisteme (calculatoare, terminale) se numesc circuite fizice.

Pentru retelele care contin un numar cit de cit semnificativ de puncte de acces, utilizarea in

exclusivitate a circuitelor fizice este total nepractica si neeconomica.

Retelele permit servicii punct-la-punct (care ar necesita circuite fizice distincte) utilizind circuitele virtuale.

Intr-un circuit virtual format intre doua puncte de acces oarecare ale retelei se folosesc atit circuitele fizice (componentele hardware aferente) cit si software-ul de retea pentru transferul de date intre punctul de acces "local" si cel de la "distanta". Astfel, circuitele virtuale permit accesul la oricare punct al retelei fara a fi nevoie de o retea completa de cabluri si componente hardware care sa conecteze fizic fiecare pereche de puncte din retea.

2.2. Retele locale de date (LAN)

Retelele locale de date (LAN-Local Area Network) difera in principal de retelele generale de date (WAN-Wide Area Network) prin distanta maxima permisa intre doua puncte ale retelei. Desigur ca aceasta definitie nu este foarte precisa, diferentele intre cele doua tipuri de retele fiind mai pronuntate si mai variate decit simpla comparatie a distantelor acoperite.

Topologii principale in LAN

Principalele tehnologii (moduri de conectare a nodurilor) utilizate in retelele locale de date sint: magistrala (bus) comuna, inel si stea (radiala), ilustrate in figura 2.

Exemple "clasice" de retele locale sint: cu magistrala comuna-Ethernet, cu topologie inel IBM-TRN, cu topologie stea (radiala) AT&T-ISN.

Dispozitive cu interfata la retea (NIU)

Un exemplu de dispozitive de interfata (puncte de acces) la o retea locala (NIU-Network Interface Unit) il constituie echipamentele "inteligente" care dispun de doua sau mai multe "porti" RS-232 (figura 3). NIU contine software de control retea care permite crearea de circuite virtuale intre portile RS-232 proprii si porti RS-232 ale altor NIU din reteaua respectiva. Din punct de vedere functional, dispozitivele de interfata de retea sint similare cu modem-urile inteligente. Ele functioneaza, ca si modem-urile inteligente, in mod comanda si in mod date. Modul comanda este utilizat pentru stabilirea si "distrugerea" circuitelor virtuale iar in modul date se vehiculeaza date intre cele doua capete ale circuitului virtual. Majoritatea tipurilor de NIU

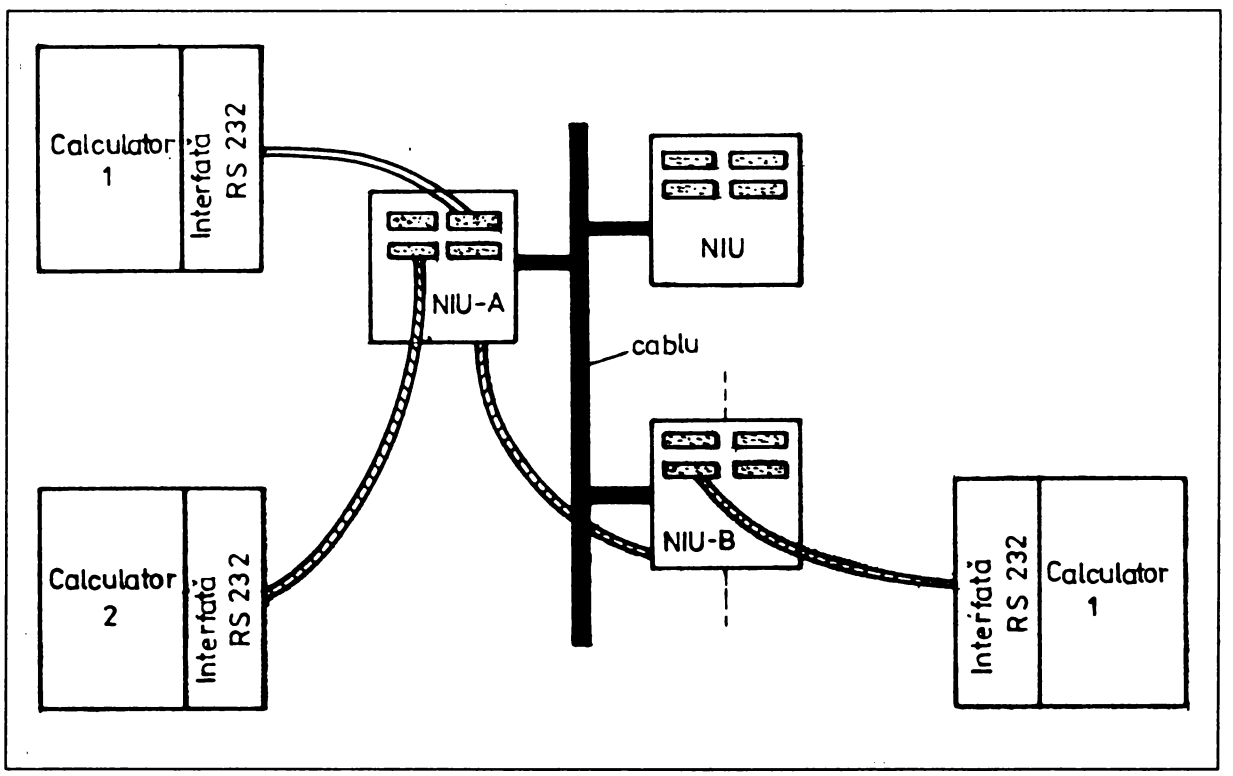


Figura 3. Retea locala de tip Ethernet (magistrala comuna)

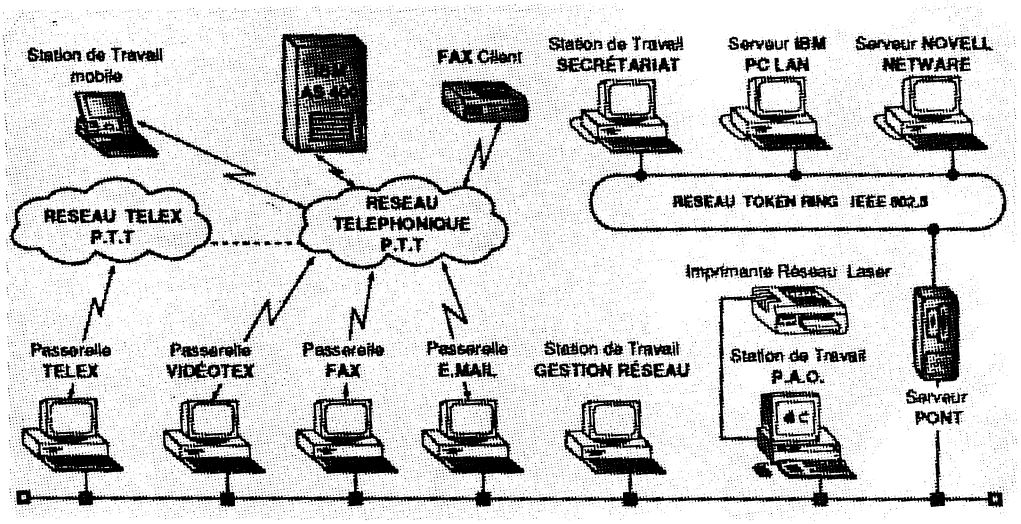


Figura 4. Retea generala de transmisie de date

impun si un control al fluxului de date RS-232 generat de calculatorul sursa.

De asemenea, se pot realiza retele locale simple si ieftine cu configuratie radiala (stea), cu calculatoare cu interfete RS-232 conectate prin cable RS-232 la un dispozitiv central de comutare. Acest dispozitiv central de comutare dispune de un software simplu de control retea care realizeaza cele doua moduri (asemanator cu un NIU pentru retele cu magistrala comuna): modul comanda (pentru realizarea circuitelor virtuale) si modul date (pentru transferul de date intre capetele circuitului virtual).

2.3. Retele generale de date (WAN)

Retele generale pentru transmisia de date, acestea acopera distante geografice mari, uneori o intreaga tara, aceasta in contrast cu retelele locale. Cu toate diferentele de marime si topologice ele sint similare in multe privinte cu retelele locale.

Exemple de retele generale: in SUA retelele Telenet, Tymnet, Accunet, DepNet deservesc majoritatea marilor orase. Din aceste orase, prin conectarea la un punct de acces la retea, se poate solicita un circuit virtual cu oricare alt punct din retea. Astfel, se poate avea acces la unele servicii de baze de date centralizate (ca, de exemplu, The Source si CompuServe).

Accesul in retelele generale

Se realizeaza, ca si in scopul retelelor locale, prin puncte de acces-dispozitive de interfata denumite pe scurt PAD (Packet Assembler/Disassembler).

Retelele generale de date folosesc de regula o forma specifica de comunicatie denumita comutare de pachete care este definita de catre standardul X.25.

Dispozitivul de acces la reteaua generala (PAD) este asemanator functional cu dispozitivul de acces la reteaua locala (NIU): este conectat la retea printr-o linie de mare viteza (19200, 48000 bps sau mai mare) si dispune de citeva porti RS-232 la care se conecteaza prin circuite telefonice la mica viteza terminale, microcalculatoare, calculatoare personale, etc. (vezi figura 4).

Dispozitivul de acces asambleaza datele asincrone sosite de la abonati in pachete care sint apoi dirijate catre destinatie pe circuitele virtuale ale retelei generale.

Ce place la un joc pe calculator?

De ce jocurile pe calculator sint atit de captivante? Si cum se poate ca aceleasi lucruri care fac jocurile captivante sa faca procesul invatarii (cu calculatorul) mai interesant si mai placut?

Pentru a raspunde la aceste intrebari, Thomas W. Marlowe de la Cognitive Instructional Science Group a realizat un studiu asupra a 100 de persoane amatoare de jocuri pe calculator, urmarind cu precadere ceea ce place in mod deosebit la aceste jocuri. Din acest studiu a desprins o serie de observatii de baza in exprimarea la nivel inalt a motivatiei educationale a programelor pe calculator.

Desi scopul cercetarii a fost acela de a da activitatilor educationale o tenta mai placuta, distractiva chiar, directiile formulate pot fi utilizate la fel de bine atit pentru realizarea de jocuri pe calculator fara profil educational cit si pentru realizarea de programe a caror utilizare sa fie mai agreabila.

Cercetarea preferintelor

Una din intrebarile cele mai interesante care s-au formulat unor subiecti, a caror virsta mergea de la cea de gradinita pina la clasele superioare, a fost, ce lipseste jocurilor nepopulare in comparatie cu cele populare?

Cel mai important factor care determina popularitatea unui joc a rezultat a fi existenta unui anumit scop bine precizat. De asemenea, afisarea scorului, efectele sonore precum si aleatorul prezinta corelatii strinse cu popularitatea jocurilor.

Jocul BREAKOUT

Se mai numeste WALL sau BRICKS si consta in distrugerea unui zid cu ajutorul unei bile care circula intre acesta si un cursor al utilizatorului.

Cercetarea realizata a indicat ca este unul dintre cele mai populare jocuri. Care este "secretul" acestui succes? Foarte multi dintre cei incintati de acest joc, atunci cind vorbesc despre el, mentioneaza scorul pe care l-au obtinut, de obicei pe cel mai mare. Este oare provocarea obtinerii celui mai inalt scor principala atractie? Este stimularea vizuala a observarii desprinderii caramizilor din zid, deci a distrugerii zidului? Sau este simpla bucurie a performantelor senzo-motoare proprii, demonstrate in manipularea cursorului (paleta)?

Desigur, mai exista si alte caracteristici ale acestui joc,dar se pare ca cele trei - scorul, spargerea caramizilor, minuirea paletei - exprima esenta jocului.

Pentru realizarea in continuare a ierarhizarii acestor caracteristici s-au construit sase versiuni diferite ale jocului. Spre exemplu, in unele versiuni, bila circula intre paleta si zid, dar nu se desprind caramizile din zid. In altele, bila nu mai este respinsa de paleta ci este "prinsa" atunci cind paleta se plaseaza in fata ei. De asemenea, numai jumatate din versiuni afiseaza un scor.

Din studierea preferintelor celor care au jucat toate versiunile, s-a desprins faptul ca desprinderea caramizilor din zid este lucrul cel mai important. Distrugerea partiala a zidului prezinta vizual atingerea scopului propus, precum si cit de aproape este jucatorul de acest scop. Astfel, zidul are scopul

de a realiza atit un efect vizual, cit si prezentarea intr-o forma grafica a scorului. De fapt structura zidului sugereaza mai multe scopuri la nivele diferite: distrugerea unei caramizi cu o valoare mai mare intr-un rind mai indepartat; distrugerea completa a primului rind.

Rezultatul demonstreaza faptul ca versiunile fara scor si fara distrugerea zidului sint in mod semnificativ dezagreate in comparatie cu celelalte versiuni. Cu alte cuvinte, versiunile care nu au scop clar, sint mai putin placute. Un exemplu ar fi cele in care se incearca mentinerea circulatiei bilei intre zid si paleta o perioada cit mai indelungata. Acelasi lucru este valabil si in cazul jocurilor cu scopuri diferite, ierarhizate pe nivele diferite. Pentru succesul unor astfel de jocuri, ele trebuie sa fie foarte clare.

Jocul DARTS (SAGEATA)

Al doilea joc studiat este destinat invatarii elevilor din clasele elementare a notiunilor de baza despre fractii. In prima versiune sint utilizate trei baloane care apar in mod aleator in locuri diferite pe o linie numerotata pe ecran, iar jucatorii incearca sa ghiceasca pozitiile lor. Ghicirea se realizeaza prin introducerea numerelor intregi impreuna cu partea fractionara. Dupa fiecare ghicire, o sageata va parcurge ecranul si se va plasa pe linie conform numarului introdus. In cazul in care s-a ghicit pozitia, balonul se va sparge, dar daca s-a gresit, va ramine pe ecran. Jucatorul va "trage" pina cind toate baloanele vor fi sparte.

La inceputul jocului este audiata o muzica de lire, iar in cazul in care cele trei baloane sint sparte din trei incercari se va audia ca recompensa un scurt cintec. Pentru a gasi caracteristicile care fac acest joc atractiv s-au construit opt versiuni diferite, inlocuind pe rind pe cele care au fost presupuse importante din punct de vedere motivational. Spre exemplu, unele versiuni au dreptunghiuri in loc de baloane, pentru marcarea locului care trebuie ghicit, altele au liniute in locul sagetilor care marcheaza o pozitie incorecta. Alte caracteristici modificate includ formele sagetilor care sparg baloanele, muzica, afisarea scorului si unele moduri de reluare. Pentru studiu fiecare jucator a avut propria lui versiune cu care s-a jucat, sau o versiune a jocului "HANGMAN", care a fost aceiasi pentru toti.

Primul lucru studiat a fost cit timp s-au jucat versiunile de DARTS in comparatie cu HANGMAN. Aceasta masura a fost corelata cu raspunsurile jucatorilor in privinta agreerii jocurilor.

Rezultatul experimentului arata o diferenta semnificativa intre baieti si fete, legat de ceea ce le-a placut mai mult la joc. Judecind in functie de timpul petrecut cu jocul respectiv, a rezultat ca baietilor le-au placut formele sagetilor care spargeau baloanele, in timp ce fetelor, aparent, le-au displacut. Autorul nu crede ca baietii au un anumit fel de fantezie, iar fetele alta, considerind ca ar fi mai bine a se lasa alegerea din partea jucatorilor, a formei celei mai placute la un anumit moment. Intelegerea unei astfel de diferente intre sexe poate fi un ajutor pentru realizarea de programe destinate, de exemplu, mai mult baietilor decit fetelor.

O concluzie care reiese de aici este aceea ca, desi este importanta in crearea programelor educationale interesante, fantezia trebuie cu grija aleasa pentru a fi receptionata cu placere.

Cum pot fi utilizate aceste rezultate pentru realizarea unor programe educationale mai placute? Autorul studiului considera ca principalele caracteristici care fac mediul instruirii mai interesant se pot grupa in trei categorii:

- competitie,
- fantezie,
- curiozitate.

Pentru ca o activitate sa fie competitiva ea trebuie sa aibe un scop a carui atingere este incerta. Din studiul realizat a reiesit ca scopul trebuie sa fie foarte clar. Mediile mult mai complexe (ca editoarele grafice sau sistemele de dezvoltare pentru limbajele de programare) trebuie realizate in asa fel incit utilizatorii sa poata genera usor scopuri cu o dificultate adecvata.

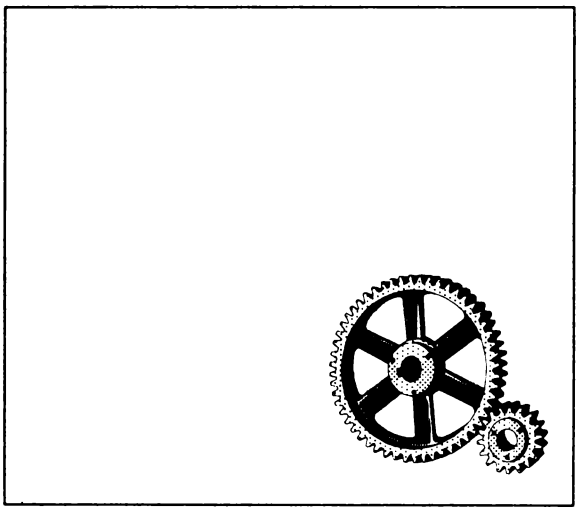
Spre exemplu, in sistemul LOGO, prin miscarea unui cursor, "broscuta", se poate desena. Activitatea acestui mediu este data de usurinta cu care copiii gindesc la lucruri pe care ar dori sa le realizeze broscuta prin miscare. Desi incepatorii sint ajutati in evaluarea unor proiecte posibile, de multe ori aleg lucruri care sint foarte dificile. Scopurile bune au de asemenea semnificatie personala. De exemplu, cele mai bune par cele practice sau fanteziste (cum ar fi ajungerea la luna cu o racheta sau desenarea unei flori intr-un tablou) spre deosebire de scopurile unei probleme.

Se recomanda ca nivelul de dificultate sa varieze, acesta putind fi:

- determinat automat,
- ales de jucator,
- determinat de nivelul oponentilor.

Nivelul de dificultate se poate determina prin pastrarea si analizarea scopului realizat precum si a vitezei de raspuns. Acestea sint foarte utile in situatii in care se cere invatarea si practicarea pentru verificarea nivelurilor deja cunoscute.

Succesul intr-un joc poate da incredere celui care-l obtine asupra capacitatilor personale. Datorita posibilitatii de insucces se recomanda ca jocurile sa aibe nivele de dificultate diferite.



In continuare se propune un joc care sa mareasca interesul in legatura cu diferitele sisteme de notare a timpului: ceasul obisnuit, afisari digitale, cuvinte in limbaj natural. Propunerea a fost facuta de Laura Gould si presupune trei forme diferite de sisteme de reprezentare care vor fi afisate pe ecran, astfel incit atunci cind un jucator schimba o reprezentare se vor schimba automat si celelalte doua. In acest joc timpul este reprezentat intr-un sistem, de exemplu ceas obisnuit si jucatorii incearca sa ghiceasca timpul in unul din celelalte sisteme, de exemplu afisarea digitala. Orice incercare nereusita va fi afisata pe ceasul obisnuit la fel ca in jocul DARTS. Jocul poate deveni mai captivant daca se includ sunete de alarma, sau daca ora afisata pe ecran nu corespunde orei oficiale.

In general un joc poate sugera jocuri similare, dar foarte diferite ca subiect in comparatie cu cel original. Spre exemplu, structura unui joc-ghicitoare poate fi folosit pentru inventarea altor jocuri care sa permita invatarea diferitelor cunostinte:

- pentru invatarea unei liste ordonate se recomanda un joc-ghicitoare in care sa se returneze un raspuns pozitional (mai sus sau mai jos);
- pentru invatarea corespondentei intre doua sisteme de reprezentare se recomanda utilizarea unui joc-ghicitoare care sa sugereze o marime in cadrul unui sistem si sa ceara corespondentului sa o introduca in celalalt sistem (asemanatoare jocurilor DARTS sau cel cu ceasurile). Astfel de jocuri pot fi utilizate pentru invatarea corespondentei cuvintelor din vocabularele a doua limbi diferite;
- pentru invatarea caracteristicilor unor tipuri din cadrul unui set (multimi) se recomanda utilizarea unui joc in care sa se ghiceasca tipul in functie de intrebarile care se pun asupra caracteristicilor. Un exemplu ar fi stabilirea diagnosticului unui pacient simulat, prin intrebari asupra simptomelor si analizelor de laborator.

Aceasta tehnica de utilizare a analogiilor structurale cu jocurile vechi pare a fi o cale serioasa in inventarea de jocuri educationale in noi zone de interes.

Intr-un anumit sens, programarea calculatoarelor este cel mai frumos joc din toate. In "jocul de programare a calculatoarelor" exista scopuri clare si, chiar mai mult, sint usor de formulat. Jocul poate fi jucat la diferite nivele de dificultate si este posibila ierarhizarea scopurilor pe nivele. In legatura cu cele de mai sus, se pot formula o serie de intrebari ca: daca va merge, cit de repede va merge, cit spatiu necesita, cit dureaza programarea, etc. Respectul fata de sine este implicat in acest "joc", iar emotiile ocazionale precum si aspectele de fantezie sint probabil implicate in controlarea deplina a comportamentului de raspuns al jucatorului.

Aparitia si inmultirea calculatoarelor atit in locuinte cit si in scoli este inevitabila, dar nu este cert ca aceste citeva noi aplicatii educationale prezentate exprima singurele posibilitati de utilizare a calculatoarelor in realizarea unei invatari mai eficiente, mai interesante si, nu in ultimul rind, mai placute. Problemele prezentate pot contribui in realizarea de jocuri pe calculator care sa fascineze si sa educe in acelasi timp.

Utilizarea unui dispozitiv mouse pe calculatoarele personale

Marcel Vladescu

Unul dintre dispozitivele de intrare cele mai populare si mai des folosite in lumea calculatoarelor personale este mouse-ul. Comoditatea utilizarii sale este binecunoscuta tuturor celor care au avut contact cu programe precum Microsoft Windows si aplicatiile scrise pentru acesta, PC Paintbrush sau XEROX Ventura Publisher. Probabil ca multi programatori au dorit sa includa in interfetele programelor lor posibilitatea folosirii acestui dispozitiv, dar s-au lovit de lipsa unei documentatii adecvate, iar dorinta lor a ramas la stadiul de simpla intentie. Acestora, precum si tuturor celor interesati si dornici sa creeze programe la fel de frumoase si de usor de utilizat ca Ventura Publisher, le oferim sprijin prin seria de articole pe care o deschidem in numarul de fata. Aceste articole vor prezenta complet, in detaliu, interfata software dintre un program de aplicatie si un mouse.

Precizam de la inceput ca este vorba despre interfata pentru Microsoft Mouse, interfata care s-a impus ca standard pentru dispozitivele mouse. Practic, toate firmele care produc astfel de dispozitive, fie le produc complet compatibile cu cele Microsoft, fie asigura un mod de lucru compatibil. Aceasta standardizare ne asigura ca un program care recunoaste si utilizeaza un mouse Microsoft va putea recunoaste si utiliza un mouse de orice tip, indiferent de firma producatoare. Interfata specificata de Microsoft s-a impus ca standard datorita numarului mare de functii pe care le ofera, permitind scrierea unor aplicatii sofisticate si pretentioase pentru calculatoare IBM PC/XT/AT, sub sistemul de operare MS-DOS. Linia de microcalculatoare profesionale IBM PS/2 propune o interfata cu facilitati superioare, adaptata pentru multitasking in contextul sistemului de operare OS/2, implementata ca o parte a BIOS-ului acestor masini. Ne propunem sa prezentam o descriere amanuntita a acesteia in numerele viitoare ale revistei, intr-un serial mai complex dedicat familiei PS/2.

In acest numar vom prezenta interfata dintre ultima versiune a driver-ului (software) de mouse, Microsoft Mouse V6.00, si un calculator IBM PC/XT/AT. In particular, vom explica modul in care software-ul de mouse interactioneaza cu calculatorul pentru desenarea pe ecran a cursorului si controlul miscarii acestuia. In numarul viitor vom publica descrierea completa a fiecarei functii in parte. Aceste functii sint legate strins de interfata prezentata mai jos. De aceea, este important sa parcurgeti cu atentie paragrafele care urmeaza, inaintea utilizarii functiilor respective in programele dumneavoastra.

MODURILE ECRAN

Modul ecran (sau modul video) este un termen binecunoscut oricarui programator pentru IBM PC. Modul video defineste rezolutia ecranului si maniera in care adaptorul video interpreteaza si afiseaza informatia din memoria ecran. Modurile ecran disponibile depind de tipul adaptorului instalat in calculator. Lista cu modurile video suportate de software-ul de mouse este prezentata in figura 1. (C = CGA, E = EGA, M = MDA, H = Hercules, P = PC3270).

ECRANUL VIRTUAL

Software-ul de mouse vede ecranul fizic al calculatorului ca pe o matrice de puncte, numita ecran virtual. Dimensiunile acestuia depind de modul video si sint prezentate in figura 1, pentru toate modurile suportate.

Driver-ul trebuie sa cunoasca in orice moment modul in care opereaza adaptorul video. El realizeaza acest lucru interceptind intreruperea 10H, prin care programele apeleaza functiile video din BIOS. La fiecare apel al functiei care stabileste

Mod video	Adaptor	Ecran virtual	Celula caracter	Biti/pixel/ plan
00	C,E,P	640x200	16x8	–
01	C,E,P	640x200	16x8	–
02	C,E,P	640x200	8x8	–
03	C,E,P	640x200	8x8	–
04	C,E,P	640x200	2x1	2
05	C,E,P	640x200	2x1	2
06	C,E,P	640x200	1x1	1
07	M,E,P	640x200	8x8	–
0D	E	640x200	16x8	2
0E	E	640x200	1x1	1
0F	E	640x350	1x1	1
10	E	640x350	1x1	1
30	P	720x350	1x1	1
–	H	720x348	1x1	1

Figura 1. Moduri video

modul video (intreruperea 10H cu registrele AH = 0 si AL = mod), driver-ul de mouse determina care sint dimensiunile ecranului virtual care trebuie utilizat. Functia 0 este singura functie din intreruperea 10H care este interceptata de catre software-ul de mouse. Daca programul utilizeaza alte metode pentru schimbarea modului ecran (de pilda, accesul direct la registrele hardware ale adaptorului video), este posibil ca mouse-ul sa fie "dezorientat", iar cursorul sa fie desenat incorect sau chiar deloc. De aceea, nu folositi metode nestandard pentru rezolvarea unor probleme (schimbarea sau interogarea modului video, modificarea paletei de culori pe adaptoare EGA sau VGA) pentru care serviciile puse la dispozitie de catre BIOS sint suficient de puternice chiar pentru cele mai pretentioase programe.

Indiferent de modul video, driver-ul utilizeaza perechi de coordonate virtuale (X, Y) pentru localizarea oricarui obiect de pe ecran. Multe functii ale driver-ului accepta coordonate virtuale ca parametri sau intorc valori care reprezinta astfel de coordonate. La apelul unor astfel de functii, asigurati-va ca parametrii pe care ii furnizati reprezinta coordonate valide pentru modul video curent. Valorile intoarse de functiile de mouse respecta intotdeauna aceasta conditie.

In modurile video grafice 06H, 0EH, 0FH, 10H, precum si pe adaptoarele Hercules, exista o corespondenta 1:1 intre punctele din ecranul virtual si pixelii din ecranul fizic. In aceste moduri sint acceptate ca parametri valori din intregul domeniu specificat in coloana "Ecranul Virtual" din figura 1.

In modurile grafice 4 si 5 ecranul fizic foloseste 2 biti pe pixel, pentru un singur plan de memorie ecran, ceea ce face ca numarul de pixeli afisabili pe orizontala sa fie redus la jumatate. Pentru a compensa aceasta, software-ul de mouse utilizeaza numai coordonate virtuale orizontale care sint numere pare. Altfel spus, numai fiecare al doilea punct din ecranul virtual corespunde efectiv unui pixel afisabil.

Modurile video 2, 3 si 7 sint moduri text, in care pe ecran pot fi afisate numai caractere. Fiecare caracter este un grup de 8x8 pixeli. Deoarece software-ul de mouse nu poate accesa pixelii individuali dintr-un caracter, el utilizeaza coordonatele pixelului din coltul stinga-sus al cutiei caracterului. Intrucit fiecare caracter reprezinta o celula de 8x8 pixeli, atit coordonatele orizontale cit si cele verticale sint multipli de 8.

In modurile text 0 si 1, la fel ca in modurile text 2, 3 si 7, pe ecran nu se pot afisa decit caractere. Fiecare caracter reprezinta o celula de 16x8 pixeli. Software-ul de mouse utilizeaza coordonatele unui singur pixel din celula caracterului. Totusi deoarece ecranul are, pe orizontala, numai jumatate din numarul de pixeli din modurile 2, 3 sau 7, software-ul foloseste numai coordonate orizontale multipli de 16.

CURSORUL

Software-ul de mouse poate gestiona trei tipuri distincte de cursoare:

- cursorul grafic, care este o matrice de pixeli care poate fi miscata peste imaginile de pe ecran;

- cursorul text software este un atribut de text, de exemplu video invers, care poate fi mutat de la un caracter la altul;
- cursorul text hardware este obtinut prin modificarea formei cursorului hardware implicit mentinut de adaptorul video pe un ecran in mod text.

Numai unul dintre aceste cursoare poate fi activ la un moment dat. Driver-ul de mouse contine functii ce permit selectarea tipului de cursor dorit, modificarea atributelor sale si a modului in care cursorul interactioneaza cu imaginile prezente deja pe ecran. Aceste functii vor fi prezentate in numarul viitor al revistei.

Driver-ul asigura in permanenta salvarea si restaurarea automata, transparenta fata de programul de aplicatie, a portiunilor de ecran peste care se deseneaza cursorul mouse-ului, la fiecare deplasare a acestuia.

In paragrafele care urmeaza sint descrise in detaliu cele trei tipuri de cursoare.

CURSORUL GRAFIC

Cursorul grafic, utilizat atunci cind modul video curent este un mod grafic, este o matrice de pixeli, astfel:

- in modurile 06H, 0EH, 0FH, 10H, 30H si Hercules este o matrice de 16x16 pixeli;
- in modurile 4 si 5 este o matrice de 8x16 pixeli.

Odata cu deplasarea mouse-ului pe suprafata de lucru, blocul de pixeli se deplaseaza pe ecran si interactioneaza cu pixelii de dedesubt. Aceasta interactiune creaza doua zone distincte in imaginea cursorului: forma efectiva ("foreground") si fondul imaginii ("background"). Modalitatea in care se va afisa imaginea cursorului este definita de doua matrice de 16x16 biti, masca de cursor si masca ecran, care pot fi stabilite prin programul de aplicatie:

- masca de ecran decide, pentru fiecare pixel in parte, daca acesta va face parte din forma efectiva a cursorului sau din fondul imaginii cursorului;
- masca de cursor determina modul in care pixelii de sub cursor contribuie la stabilirea culorii acestuia.

Programul de aplicatie poate modifica aceste doua matrice si transmite ca parametri functiei 9 a driver-ului. Interactiunea dintre pixelii de pe ecran si cele doua masti in scopul crearii cursorului difera de la un mod video grafic la altul. In modurile 06H, 0FH, 30H si Hercules, fiecare bit din cele doua masti corespunde direct unui pixel de pe ecran. In modurile 4 si 5 fiecare pereche de biti corespunde unui pixel.

Pentru generarea cursorului, driver-ul de mouse opereaza asupra datelor din memoria ecran, care determina culoarea pixelilor din zona cursorului. Mai intii, software-ul efectueaza o operatie logica AND intre masca de ecran si cei 256 de biti din memoria ecran corespunzatori zonei cursorului. Rezultatul acestei operatii este supus apoi unei operatii XOR cu masca de cursor.

In modurile EGA, 0EH si 10H, care folosesc patru plane de memorie ecran, fiecare bit din mastile de cursor si ecran corespund unui pixel de pe ecran. Fiecare plan de memorie are masti de cursor si ecran proprii. Cei patru biti care rezulta din operatiile descrise anterior constituie adresa, in paleta de culori, a culorii finale a cursorului. In aceste moduri video, driver-ul de mouse seteaza toti bitii registrului "Write-Mask" al adaptorului EGA la 1. De aceea, dupa ce programul apeleaza functia 9 pentru schimbarea formei cursorului, pixelii din imaginea cursorului pot fi numai negri, transparenti sau inversati. Aceasta versiune a driver-ului de mouse nu suporta alte culori pentru cursor in afara de negru sau alb stralucitor.

Referirile la pozitia cursorului grafic in cadrul descrierii functiilor driver-ului au in vedere pozitia punctului de referinta al cursorului. Acest punct se defineste trimitind coordonatele sale functiei 9. Valorile acestor coordonate pot fi cuprinse intre -16 si 16, dar, in modurile 4 si 5, coordonata orizontala trebuie sa fie un numar par. In toate modurile grafice, aceste coordonate sint relative la coltul din stinga sus al blocului cursorului.

CURSORUL TEXT SOFTWARE

Acest cursor poate fi utilizat numai cind adaptorul video lucreaza intr-unul dintre modurile text.

Cursorul schimba modul in care caracterele apar pe ecran. Spre deosebire de cursorul grafic, cursorul text nu are, de regula, o forma proprie. In schimb, el modifica atributele (culoare, intensitate, subliniere)

corespunzind caracterului din pozitia curenta. Cursorul text poate avea si o "forma" proprie si anume, unul dintre cele 256 de caractere ale setului ASCII extins (IBM). Cele mai multe aplicatii se limiteaza, insa, la modificarea atributelor din pozitia curenta.

Efectul produs de cursor asupra caracterelor peste care trece este definit de doua numere de 16 biti, numite masca de cursor si masca ecran:

- masca ecran specifica atributele caracterului de sub cursor care vor fi folosite pentru generarea

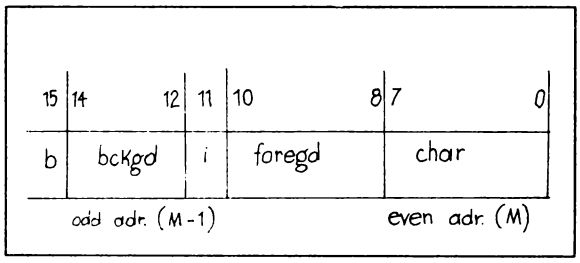


Figura 2. Caracter/atribut

cursorului;
- masca de cursor precizeaza cum trebuie modificate atributele caracterului pentru a obtine cursorul.

Pentru a crea cursorul, software-ul de mouse opereaza asupra celor doi octeti din memoria ecran care corespund caracterului din pozitia cursorului (codul caracterului si atributul acestuia). Mai intii se executa o operatie AND intre masca de ecran si cei 16 biti din memoria ecran. Apoi se efectueaza o operatie XOR intre masca de cursor si rezultatul operatiei AND, rezultatul final constituind cursorul, care se depune in memoria ecran.

Cei 16 biti din memoria ecran care descriu un caracter au configuratia din figura 2. Mastile de cursor si ecran contin aceleasi cimpuri ca cele aratate in figura mai sus mentionata. Valorile acestor cimpuri definesc noile atribute ale caracterului de sub cursorul de mouse (eventual si codul caracterului, tehnica destul de rar intrebuintata, dupa cum am mai spus). Iata si citeva exemple simple:

- Pentru a crea un cursor care inverseaza culoarea caracterului si culoarea fondului caracterului se vor utiliza mastile:

01110111 11111111 (binar) = 77FF (hexa), pentru masca ecran;

01110111 00000000 (binar) = 7700 (hexa), pentru masca de cursor.

Acesta este cursorul text software implicit, disponibil imediat dupa initializarea mouse-ului prin apelul functiei 0.

- Pentru a crea un cursor in care culorile caracterului din pozitia curenta ramin nemodificate dar acest caracter este inlocuit cu caracterul '$' (codul ASCII 24H) vom folosi:

11111111 00000000 = 0FF00h, pentru masca ecran;

00000000 00100100 = 0FF24h, pentru masca de cursor.

Intr-un program masca de cursor si masca ecran se stabilesc apelind functia 10 cu valorile acestor masti ca parametri.

Ori de cite ori o functie a driver-ului se refera la pozitia unui cursor text, este vorba despre coordonatele caracterului de sub cursor. Cursoarele text, spre deosebire de cele grafice, nu au un punct de referinta.

CURSORUL TEXT HARDWARE

Cursorul text hardware este un alt tip de cursor care poate fi utilizat numai in moduri video text.

Acest cursor este, de fapt, chiar cursorul adaptorului video (cursorul care apare pe ecran imediat dupa pornirea calculatorului). Software-ul de mouse permite modificarea dupa dorinta a marimii acestui cursor. Clipirea cursorului text hardware nu poate fi dezactivata sau modificata in nici un fel pe calculatoarele din familia IBM PC.

Cursorul are o latime de 8 pixeli si o inaltime de pina la 32 de linii. Linia superioara este linia 0. Numarul maxim efectiv valid, de linii, depinde de adaptorul video instalat in sistem. Un program poate modifica dimensiunea verticala a cursorului (limitele inferioara si superioara a grupului de linii aprinse) prin apelarea functiei 10 a driver-ului de mouse. Aceasta functie are acelasi efect asupra cursorului ca si functia 01H din BIOS-ul video (intreruperea 10H cu registrul AH = 01H). In plus, insa, functia 10 permite utilizarea acestui cursor

drept cursor de mouse, care isi schimba pozitia pe ecran odata cu deplasarea mouse-ului pe masa sau pe alta suprafata plana. Parametrii primiti (limitele inferioara si superioara ale cursorului) nu sint verificati de catre functia 10. Valorile trebuie sa se incadreze in limitele permise de fiecare adaptor video in parte, conform listei de mai jos:

- pe adaptoare CGA, cursorul are 8 linii, numerotate de la 0 la 7;
- pe adaptoare MDA, cursorul are 14 linii, numerotate de la 0 la 13;
- pe adaptoare EGA conectate la un IBM Color Display, cursorul are 8 linii; daca EGA este conectat la un IBM Enhanced Color Display, cursorul are 14 linii.

Valori invalide pentru acesti parametri pot provoca disparitia cursorului sau miscarea sa necontrolata pe tot ecranul. Adaptorul video al calculatoarelor PC 3270 nu suporta decit cursoare bloc (complet aprinse), parametrii trimisi functiei 10 fiind ignorati.

BUTOANELE MOUSE-ULUI

Un mouse poate avea intre unu si trei butoane. Cele mai multe programe care suporta un mouse, inclusiv programe "mari", precum Microsoft Windows, AutoCAD sau Ventura, sint scrise pentru limita inferioara si este bine sa incercati acelasi lucru in programele dumneavoastra. Pe mouse-uri cu mai multe butoane se utilizeaza butonul din stinga. Dispozitivele originale fabricate de catre Microsoft (si interfata software a acestora) au intotdeauna doua butoane.

Functiile 5 si 6 ale driver-ului de mouse intorc starea butoanelor dispozitivului si contorizeaza numarul de apasari si eliberari ale butoanelor. Starea butoanelor este intoarsa ca un numar intreg, in care numai primii doi biti contin informatie, indicind starea fiecaruia dintre cele doua butoane. Bitul 0 (cel mai putin semnificativ) reprezinta starea butonului din stinga, bitul 1 pe cea a butonului din dreapta. Un bit cu valoarea 1 reprezinta un buton "apasat", valoarea 0 corespunzind starii "eliberat" a butonului. Software-ul incrementeaza cite un contor de fiecare data cind un buton este apasat sau eliberat. Valoarea unui contor este adusa la zero la apelarea functiei 0, de initializare a mouse-ului, sau dupa fiecare citire a continutului sau.

CUM SE MASOARA MISCAREA UNUI MOUSE?

Miscarile mouse-ului sint traduse in valori care exprima marimea, directia si durata acestor miscari. Aceste valori sint date in unitati de masura specifice mouse-ului, numite, in limba engleza, mickeys (nu cred ca puteti traduce cumva acest termen in limba romana!). Un mickey este aproximativ echivalent cu 1/200 inch. La miscarea mouse-ului pe suprafata de lucru, hardware-ul transmite driver-ului software drumul parcurs pe verticala si orizontala, in mickeys. Software-ul transforma aceste miscari in deplasari pe ecran ale cursorului. Numarul de pixeli parcurs de cursor pe ecran nu corespunde neaparat unu la unu cu numarul de mickeys parcursi de bila mouse-ului. Driver-ul defineste sensibilitatea mouse-ului ca fiind numarul de mickeys care produc o deplasare de 8 pixeli a cursorului. Programul poate modifica sensibilitatile mouse-ului, nu neaparat identice pe orizontala si verticala, prin apelul functiei 15. Numerele specificate pot avea orice valoare intreaga intre 1 si 32767.

STAREA CURSORULUI

Software-ul de mouse gestioneaza o variabila care precizeaza starea cursorului: vizibil sau invizibil. Valoarea acestei variabile este intotdeauna mai mica sau egala cu zero:

- daca valoarea variabilei este egala cu zero, cursorul este vizibil;
- daca valoarea este mai mica decit zero, cursorul nu este vizibil.

Programele nu pot modifica direct aceasta variabila. Pentru modificarea valorii sale exista doua functii speciale ale driver-ului. Functia 1 incrementeaza valoarea, iar functia 2 o decrementeaza. Valoarea initiala a acestei variabile, dupa initializare, este -1. Asadar, imediat dupa initializarea mouse-ului, cursorul acestuia este invizibil, fiind necesar un apel al functiei 1 pentru afisarea sa.

Un program poate apela atit functia 1, cit si functia 2, de cite ori doreste, dar, pentru fiecare apel al functiei 2 este necesar un apel al functiei 1, pentru a restabili valoarea precedenta a variabilei. Atunci cind starea cursorului este "vizibil", apelarea in conrinuare a functiei 1 nu mai are nici un efect, nici asupra cursorului, nici asupra valorii variabilei de stare, care nu poate fi incrementata peste zero. In aceasta situatie, un singur apel al functiei 2 este

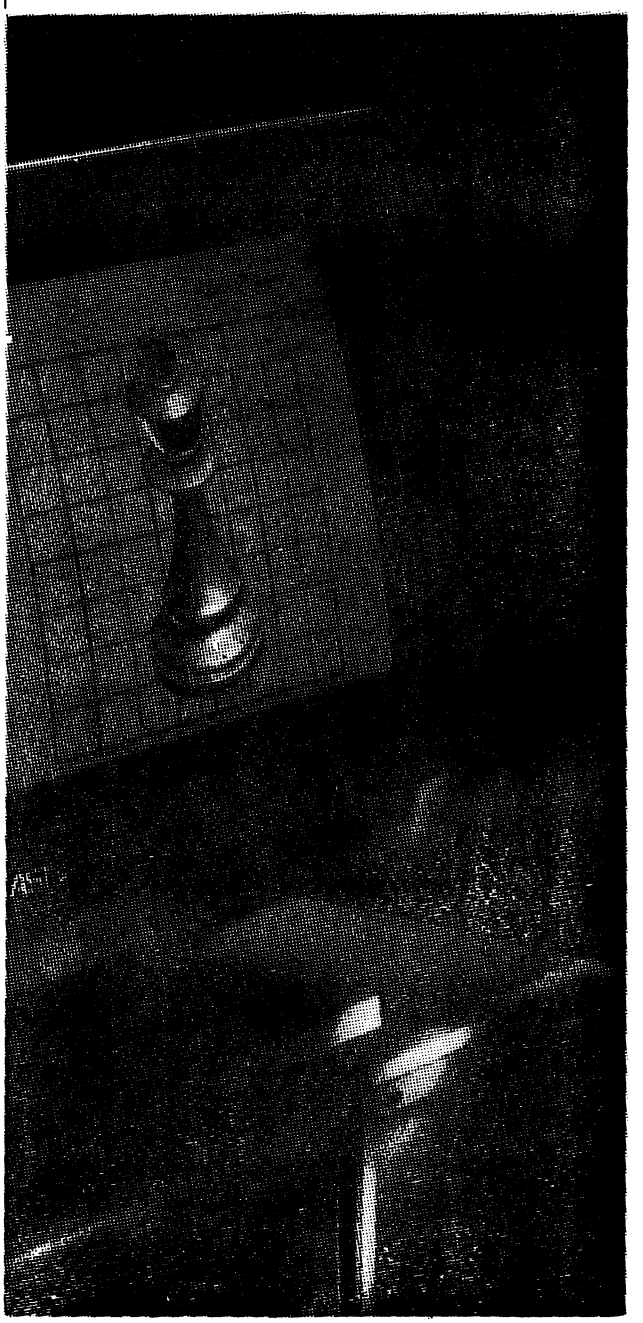
intotdeauna suficient pentru a provocà disparitia cursorului.

Scopul introducerii acestei variabile de stare este de a facilita actualizarea unor imagini pe ecran fara compromiterea imaginii cursorului de mouse sau a logicii interne de salvare si restaurare a acestuia. Secventa de utilizare este foarte simpla:

...
< apel functia 2 >
< orice operatii care modifica ceva pe ecran >
< apel functia 1 >
...

In acest fel se garanteaza invizibilitatea cursorului, deci integritatea imaginii sale, in timpul efectuarii unor operatii care modifica ecranul (de exemplu, deseneaza ceva) si, de asemenea, restaurarea starii de vizibilitate a cursorului dupa terminarea operatiilor, chiar daca acestea contin, imbricate, si alte secvente ca aceea de mai sus.

Incheiem aici aceasta prezentare a termenilor, urmind ca in numarul viitor al revistei sa incepem publicarea specificatiilor tehnice ale fiecarei functii in parte, impreuna cu comentarii si exemple de utilizare.



Intre programare si joc. Calare pe tabla de sah.

In cele ce urmeaza va propunem spre rezolvare urmatoarea problema: sa se parcurga cu calul intreaga tabla de sah, in asa fel, incit fiecare patrat sa fie atins o data si numai o data. Vom prezenta doua variante de rezolvare a problemei. Ambele variante sint recursive, de aceea, am folosit un limbaj de implementare adecvat,Pascal.

In prima varianta, singurul criteriu de alegere a unei mutari este acela de a se gasi o pozitie libera care poate fi atinsa prin mutarea calului din pozitia lui precedenta. Urmind aceasta tactica, la un moment dat s-ar putea sa ne blocam, adica sa nu mai existe nici o posibilitate de mutare a calului. In acest caz trebuie sa revenim la o pozitie anterioara si sa cautam o alta posibilitate de mutare.

Listing: CAL1.PAS

```
PROGRAM call;
USES crt;

TYPE
     pasiT=array[1..8,1..2] of integer;
CONST
     N=5;
     pasi:pasiT=((-2,-1),(-1,-2),(1,-2),(2,-1),
                 (2,1),(1,2),(-1,2),(-2,1));
VAR
     contorP,OrdP,I,J:integer;
     tabla:array[1..N,1..N] of boolean;

PROCEDURE Pas(I,J:integer);
VAR
     Pasul:integer;
BEGIN
     IF (I in [1..N]) and (J in [1..N])
          and (contorP < N*N)
          and not(tabla[i,j]) THEN
          BEGIN
               contorP:=contorP+1;
               tabla[I,J]:=true;
               Pasul:=1;
               REPEAT
                    Pas(I+Pasi[Pasul,1],J+Pasi[Pasul,2]);
                    Pasul:=Pasul+1
               UNTIL (Pasul > 8) or (contorP >= N*N);
               IF contorP < N*N THEN
                    BEGIN
                         contorP:=contorP-1;
                         tabla[i,j]:=false
                    END
```

Listing: CAL1.PAS (continuare)

```
                ELSE
                     BEGIN
                          GotoXY(3*I,3*J);
                          Writeln(OrdP:2);
                          OrdP:=OrdP-1
                     END
          END
END;

BEGIN
     FOR I:=1 TO N DO
          FOR J:=1 TO N DO
               tabla[I,J]:=false;
     contorP:=0;
     OrdP:=N*N;
     Pas(1,1)
END.
```

Programul sursa, scris in Turbo Pascal versiunea 5.00, este redat in listingul alaturat, CAL1.PAS.

Strategia de mai sus se dovedeste ineficienta, deoarece la table de dimensiuni mari ea necesita incercarea unui numar foarte mare de variante.

Teoretic, s-a demonstrat ca daca urmatoarea mutare se face la acel patrat de la care numarul variantelor posibile de a continua parcurgerea este minim, aceasta tactica conduce la un algoritm deosebit de eficient. Practic solufia este obtinuta printr-o singura parcurgere a tablei, fara sa mai fie

Listing: CAL2.PAS

```
PROGRAM cal2;
USES crt;

TYPE
     pasiT=array[1..8,1..2] of integer;
CONST
     N=8;
     pasi:pasiT=((-2,-1),(-1,-2),(1,-2),(2,-1),
                 (2,1),(1,2),(-1,2),(-2,1));
VAR
     contorP,OrdP,Pasul,Variante,minP,
     minI,minJ,I,J:integer;
     tabla:array[1..N,1..N] of boolean;

FUNCTION NrPasi(Rind,Col:integer):integer;
VAR
     Posib,I:integer;
BEGIN
     I:=1;
     Posib:=0;
```

Listing: CAL2.PAS (continuare)

```
     IF (Rind in [1..N]) and (Col in [1..N])
          and not tabla[Rind,Col] THEN
          IF contorP=N*N-1 THEN
               Posib:=1
          ELSE
               REPEAT
                    IF (Rind+Pasi[I,1] in [1..N])
                         and (Col+Pasi[I,2] in [1..N])
                         and not Tabla[Rind+Pasi[I,1],Col+Pasi[I,2]]
                         THEN
                         Posib:=Posib+1;
                    I:=I+1
               UNTIL I > 8;
     NrPasi:=Posib;
END;

PROCEDURE Pas(I,J:integer);
BEGIN
     contorP:=contorP+1;
     tabla[I,J]:=true;
     IF contorP < N*N THEN
          BEGIN
               Pasul:=1;
               minP:=8;
               REPEAT
                    variante:=NrPasi(I+Pasi[Pasul,1],J+Pasi[Pasul,2]);
                    IF (variante < minP)
                         and (variante > 0) THEN
                         BEGIN
                              minP:=variante;
                              minI:=I+pasi[Pasul,1];
                              minJ:=J+Pasi[Pasul,2]
                         END;
                    Pasul:=Pasul+1
               UNTIL (Pasul > 8) or (minP=1);
               Pas(minI,minJ)
          END;
     GotoXY(4*I,3*J);
     Writeln(OrdP:2);
     OrdP:=OrdP-1;
END;

BEGIN
     FOR I:=1 TO N DO
```

nevoie de a reveni la un pas anterior facut. Folosind acest algoritm, va prezentam a doua varianta de rezolvare a problemei, programul CAL2.PAS.

Performantele celor doua variante de programe sint mult diferite. Rezolvind problema pentru o tabla de dimensiuni 5x5, prima varianta dureaza o secunda in timp ce, folosind al doilea program, rezultatul se obtine in 17 milisecunde. Pentru o tabla

Listing: CAL2.PAS (continuare)

```
          FOR J:=1 TO N DO
               tabla[I,J]:=false;
     contorP:=0;
     OrdP:=N*N;
     Pas(1,1)
END.
```

de 7x7, folosind prima varianta, rezultatul se obtine in 4 minute si 30 secunde, pentru table mai mari timpul crescind exponential. Folosind al doilea program pentru o tabla de 20x20, rezultatul se va obtine in numai 4 secunde. Deci sint evidente avantajele celui de al doilea algoritm prezentat..

Prezentam alaturat si o solutie pentru tabla de dimensiuni 20x20.

```
  1  40  87  82   3  42  91 102   5  44 109 114   7  46 127 120   9  48  51 122
 88  81   2  41  90 101   4  43 108 113   6  45 138 119   8  47 126 121  10  49
 39  86  89 100  83  92 179 112 103 110 167 118 115 152 137 128 131  50 123  52
 80  95  84  93 214  99 104 107 178 185 116 169 166 139 130 151 136 125 132  11
 85  38 215  98 105 180 213 198 111 168 177 186 117 170 153 140 129 150  53 124
 96  79  94 247 216 223 106 181 212 197 184 165 176 187 156 135 154 141  12 133
 37 250  97 224 229 246 217 222 199 182 211 196 189 164 171 146 157 134 149  54
 78 225 248 251 254 263 230 245 218 221 200 183 210 175 188 155 172 147 142  13
249  36 255 228 261 252 285 264 231 244 195 220 201 190 163 174 145 158  55 148
226  77 260 253 286 301 262 243 284 219 232 209 194 235 202 191 162 173  14 143
 35 256 227 360 259 310 287 300 265 242 283 268 233 208 193 236 203 144 159  56
 76 359 258 311 390 361 302 309 288 299 266 241 282 269 234 207 192 161 204  15
257  34 387 368 365 312 389 362 303 308 289 298 267 240 281 270 237 206  57 160
358  75 366 395 388 391 364 313 382 315 304 307 290 297 318 239 292 271  16 205
 33 398 369 386 367 396 383 392 363 306 339 316 333 280 291 296 319 238 293  58
 74 357 374 397 394 385 372 379 314 381 348 305 340 317 334 321 330 295 272  17
355  32 399 370 373 378 393 384 349 344 341 338 279 332 329 326 273 320  59 294
 70  73 356 375 400 371 350 345 380 347 278 343 328 335 274 331 322 325  18  21
 31 354  71  68  29 352 377  66  27 342 337  64  25 276 327  62  23  20 323  60
 72  69  30 353 376  67  28 351 346  65  26 277 336  63  24 275 324  61  22  19
```

Editoare si fonturi (II)

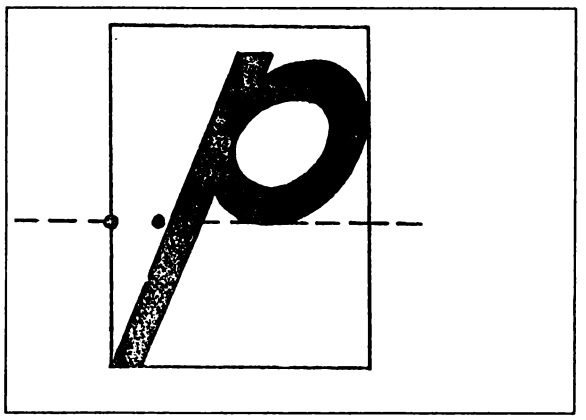
Ion Paraschiv si Tiberiu Spircu

Fonturi

Sa ne amintim ca am notat cu A o multime de semne grafice si, in grafica prin puncte (raster), am considerat ca un semn grafic este determinat de sase caracteristici:

WIDTH, ACTWID, ASC, DESC, HEIGHT, BITMAP.

In arta tiparului aspectul estetic al scrisului este obtinut si prin suprapunerea partiala a semnelor (asa-numitul *kerning*); acest efect este obtinut in grafica raster prin diferentierea caracteristicii ACTWID de WIDTH. Mai exista o metoda, anume aceea de a deplasa lateral (spre stinga sau spre dreapta) originea semnului, inainte de a fi trasat. Apare astfel o alta caracteristica a unui semn, anume aceasta deplasare (pozitiva spre stinga, negativa spre dreapta), numita OFFSET.



Referindu-ne la aspectul estetic, sa mentionam ca se mai practica metoda deplasarii laterale a unor semne atunci cind urmeaza anumitor altor semne, as-numitul *pair-kerning* (in figura alaturata este descris exemplul semnului "A" ce urmeaza semnului "V").

Nu vom insista asupra acestei metode, a kerning-ului in perechi. Pentru a incheia aceste consideratii preliminare, sa mentionam ca semnul vid va fi considerat a fi acel semn care are matricea de puncte (BITMAP) formata doar din puncte albe, dimensiunile de descriere minim posibile, iar ACTWID = OFFSET = 0.

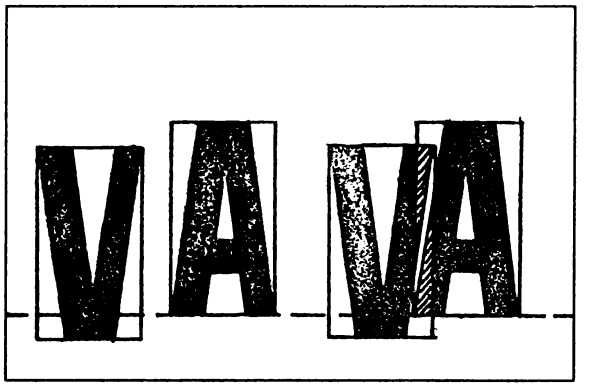
In mod obisnuit scriem insirind semne de la stinga la dreapta, in rinduri; la terminarea unui rind continuam sa scriem pe rindul urmator, de dedesubt, pina la epuizarea foii de hirtie. Aspectul estetic al scrisului trebuie sa tina seama atit de alinierea semnelor de pe un rind, cit si de distanta dintre rinduri.

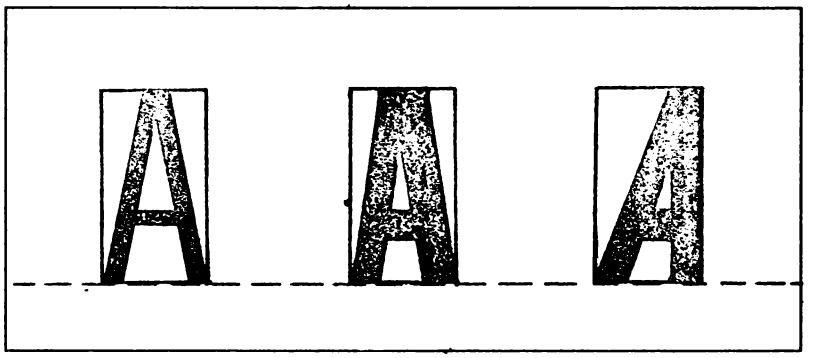
Am vazut ca fiecare semn grafic (din multimea A) este descris tinind seama de o *linie de baza*; aceasta linie este considerata, de obicei, comuna tuturor semnelor aflate pe un rind, iar distanta dintre rindurile consecutive este masurata de fapt intre liniile de baza corespunzatoare.

Uneori aceasta linie de baza este inlocuita cu o *linie de mijloc*, dar principiul este acelasi.

Se obisnuieste uneori, pentru scoatere in evidenta, ca anumite semne sa fie subliniate; linia de subliniere are o anumita grosime si se afla la o anumita distanta fata de linia de baza a semnului subliniat.

Scoaterea in evidenta a unor semne poate fi facuta si prin alte citeva efecte speciale, dintre care mentionam ingrosarea si inclinarea. Ingrosarea (**bold**) poate fi imaginata ca fiind obtinuta prin suprapunerea a doua exemplare ale aceluiasi semn, ale caror origini se afla la o anumita distanta una de alta. *Inclinarea* are loc prin deformarea specifica a semnului.





Ce este un font si prin ce se caracterizeaza el? In primul rind, introducem o functie imagine

imag: caracter ⟶ semn din A

netriviala (in sensul ca are ca valori si semne nevide) si sa ne amintim ca am ordonat total caracterele, punindu-le in corespondenta biunivoca cu numerele naturale cuprinse intre 0 si 255.

Exista evident un prim caracter, sa-l notam LOWCHR, a carui imagine nu este semnul vid; de asemenea, exista un ultim caracter, fie acesta HIGCHR, a carui imagine nu este semnul vid. Avem deci

$0 \leq \text{LOWCHR} \leq \text{HIGCHR} \leq 255.$

Sa ne indreptam atentia catre semnele nevide care sint valori ale functiei imagine. Fiecare dintre aceste semne are un *ascender* ASC si un *descender* DESC; putem alege un numar, sa-l notam ASCMAX, care sa depaseasca toate aceste ascender-e, si un alt numar, DESMAX, care sa depaseasca toate aceste descendere.

Ne putem impune si abateri maxime fata de linia de baza, fie acestea TOPMAX si BOTMAX, cu obligatia evidenta, daca vrem ca rindurile scrise sa nu se suprapuna, sa avem TOPMAX ≤ ASCMAX iar BOTMAX ≤ DESMAX.

Ne putem impune si o distanta standard intre rindurile consecutive, fie aceasta notata INTLIN.

Un *font* (abstract) este un ansamblu format dintr-o functie

imag: caracter ⟶ semn din A

(ce atrage dupa sine si precizarea lui LOWCHR si HIGCHR) impreuna cu citiva parametri numerici generali, de tipul

ASCMAX, DESMAX, TOPMAX, BOTMAX, INTLIN

si parametri specifici fiecarui semn grafic (nevid) descris. Fiecare font (abstract) mai poate avea si o denumire, fie aceasta DENFON.

In mod concret, fonturile se prezinta sub forma unor fisiere ce contin aceste informatii. Ca orice fisier, fisierele-font trebuie sa poarte un nume, alcatuit din doua parti: denumirea (fisierului, nu a fontului!) si extensia. De asemenea, aceste fisiere-font au o lungime (exprimata in octeti) care uneori se gaseste explicit in fisier. Mai pot apare, la inceputul fisierului, octeti de identificare, care sa indice ca fisierul este un fisier-font, precum si informatii neutre (indicatii asupra firmei care a creat fontul, asupra versiunii etc.).

Aparitia fonturilor este legata de aparitia mijloacelor concrete de redare a informatiei plane (a semnelor grafice): monitoare, imprimante matriciale mecanice si imprimante laser. Din punct de vedere istoric, mai intii au aparut monitoarele grafice, si in consecinta au aparut fonturile-monitor tip IBM pe care le vom prezenta in acest articol.

Aceste fonturi au fost folosite si la imprimantele matriciale EPSON (cu densitate scazuta a punctelor). Odata cu aparitia imprimantelor cu densitate mai mare au aparut si alte tipuri de fonturi.

Imprimantele laser fac parte din familia imprimantelor matriciale (prin puncte); ele "vad" o foaie de hirtie (format A4) ca o retea patratica de 3600x2550 de puncte, fiecare dintre aceste puncte putind fi innegrit independent de celelalte. Mai precis, densitatea punctelor este de 300 pe inch (1 inch = 2,54cm) la imprimantele XEROX 4045, Hewlett-Packard LaserJet Plus, EPSON GL300. De curind au aparut si imprimante laser asigurind densitatea de 600 de puncte pe inch.

In numerele urmatoare vom prezenta fonturile-imprimanta pentru HP LaserJet Plus si pentru XEROX 4045.

Desene

Oricine stie ca acum sint rare cartile in care informatia este prezentata doar cu litere, fara desene. Editarea cu ajutorul calculatorului, deci

programele-editoare, trebuie sa poata manevra desene, prezentate evident sub forma unor fisiere.

Un fisier-desen poate fi de unul din urmatoarele doua tipuri:

- de tip bitmap, insotit de informatii privind numarul de linii si numarul de coloane, in care caz pentru folosirea sa mai este necesara doar indicarea locului pe pagina in care va fi plasat desenul;
- de tip comenzi, interpretabile de catre program sau de catre dispozitivul de afisare.

Primul tip are dezavantajul ocuparii unui spatiu mare de memorie; astfel, pentru fiecare centimetru patrat de imagine reprodusa cu o imprimanta laser este nevoie de circa 16 KB de memorie. De aceea, in special pentru desenele tehnice, este recomandabil sa se foloseasca fisiere de al doilea tip.

Apare insa problema compatibilitatii, caci fiecare tip de aparat de redare isi are propriile sale reguli de interpretare. De aceea s-a pus problema standardizarii unui asa-numit meta-fisier-desen (de comenzi), din care, prin "traducere", sa se obtina fisierul-desen corespunzator aparatului de redare aflat la dispozitie.

Conform standardului GKS (Graphical Kernel System), principalele comenzi ce apar intr-un meta-fisier sint cele de:

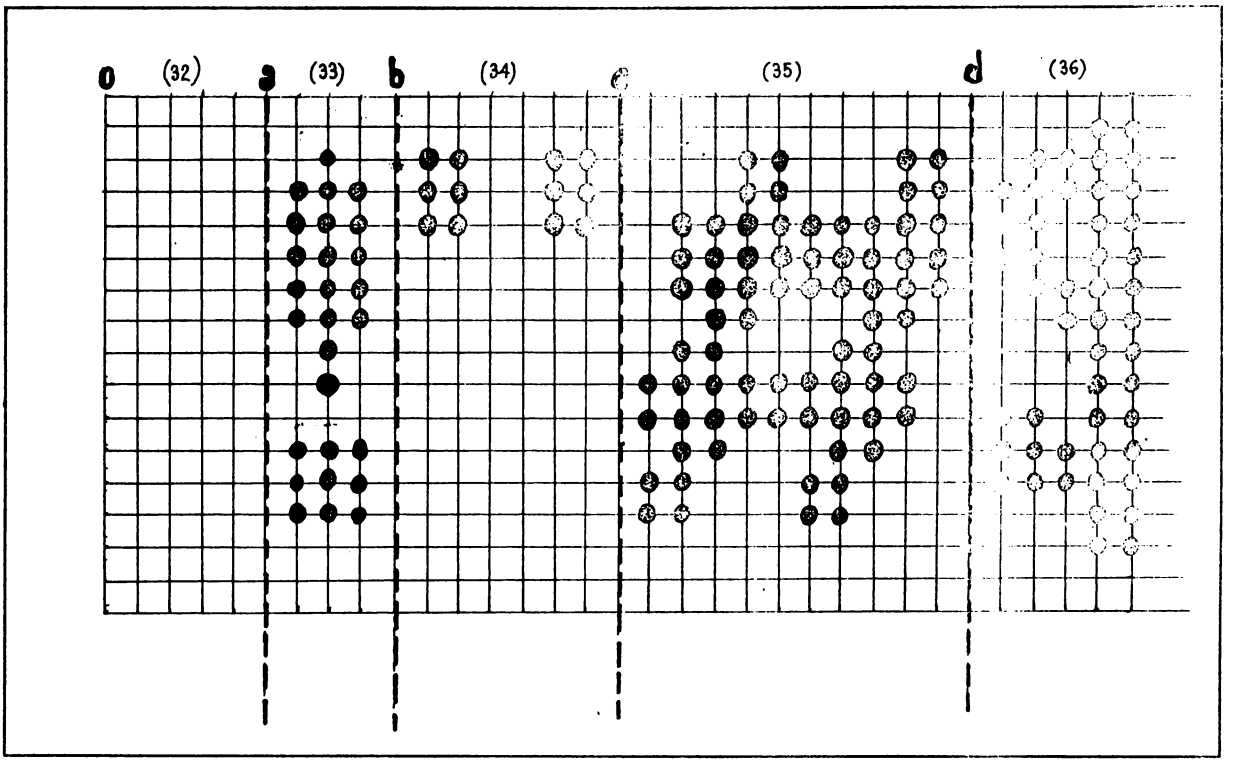
- marker (simbol "punctual");
- linie (simbol "liniar");
- celula (simbol "plan");
- text.

Nu insistam asupra standardelor grafice, problema lor urmind a fi tratata ulterior.

Descrierea unui fisier-font-monitor de tip IBM

Aceste fisiere au, de regula, o denumire ce incepe cu IBM... iar extensia indica modul grafic de lucru al monitorului, de exemplu CGA sau EGA. Pot fi intilnite si denumiri ce incep cu EPS... si au extensii EPS. Cit despre continut, sa-l explicitam octet cu octet.

0...1	numar de identificare a fontului (care, de regula, este legat de un anumit standard tipografic);
02...03	marimea standardizata, legata de traditia tipografica;
04...35	DENFON, denumirea fontului respectiv; aceasta denumire este in strinsa legatura cu traditia tipografica. Astel, vom putea intilni denumirile "Swiss" (legata de stilul literelor sans serif, sau Helvetica), "Dutch" (legata de stilul literelor tip Times) sau "Symbol" (atunci cind in font sint descrise semne speciale, de exemplu simboluri matematice);
36...37	LOWCHR
38...39	HIGCHR
40...41	TOPMAX
42...43	ASCMAX
44...45	abaterea liniei de baza fata de linia de mijloc; poate fi considerata egala cu |ASCMAX-DESMAX|/2;
46...47	DESMAX
48...49	BOTMAX (aceste marimi sint exprimate in pixeli);
50...51	latimea maxima a semnelor grafice descrise, exprimata in pixeli, ca si celelalte marimi dimensionale;
52...53	latimea maxima a unei celule, exprimata in pixeli. O celula este formata din bitmap-ul unui semn plus spatiul alb dintre semne;
54...55	OFFSET
56...57	un offset "la dreapta" semnului, analog cu cel "la stinga";
58...59	deplasarea laterala pentru ingrosare, exprimata in pixeli;
60...61	distanta pina la linia de subliniere, in cazul semnelor subliniate;
62...63	masca pentru obtinerea nuantei gri in loc de negru. Aceasta masca este de obicei 'UUUU', adica o secventa de biti alternativ albi si negri;
64...65	masca de gri utilizata in cazul inclinarii semnelor;
66...71	nefolositi
72...75	indicator (pointer) spre tabela ce contine latimile semnelor descrise; in cazul nostru are valoarea 88;
76...79	indicator (pointer) spre matricea de puncte, BITMAP; fie valoarea sa A;
80...81	lungimea in octeti a matricei de puncte;
82...83	inaltimea in pixeli a matricei de puncte;
84...87	nefolositi



88...(A-1) tabela latimilor semnelor descrise. In aceasta tabela, fiecarui caracter dintre LOWCHR si HIGCHR i se rezerva çite doi octeti. In acestia se inscrie latimea cumulata a tuturor semnelor anterioare lui, inclusiv propria-i latime (marimile 0, a, b, c,... din figura 2);

A...EOF matricea de puncte (a se vedea figura 2). Aceasta matrice este descrisa intr-un mod special; sint alipite toate BITMAP- urile semnelor, iar descrierea continutului global este data linie dupa linie.

Concepte ale programarii orientate catre obiect

Constantin Mihai si Marcel Vladescu

Notiunea de programare orientata catre obiect (in engleza, Object Oriented Programming, OOP) ocupa din ce in ce mai mult loc in literatura de specialitate pentru a putea fi ignorata. Aparitia de articole si produse care se bazeaza pe acest concept este accelerata in ultimul timp. Marea masa a programatorilor insa, manifesta nesiguranta in imbratisarea noilor concepte. Motivul principal este lipsa de familiaritate cu acestea.

Multe discutii in jurul programarii orientate catre obiect incep prin necunoasterea subiectului si sfirsesc prost din acest motiv. Cu o intelegere neclara a ceea ce inseamna "obiect", notiuni ca "mostenire" sau "ierarhie de clase" sint cauze pierdute.

Lucrul esential pe care ar trebui sa-l desprindeti din orice discutie despre programarea orientata catre obiect este ca aceasta va poate fi de un real folos in dezvoltatea aplicatiilor de mari dimensiuni. Va va fi mult mai usor sa dezvoltati asemenea aplicatii cu ea decit fara ea.

Ambele categorii de limbaje, procedurale sau orientate catre obiect, manipuleaza date si cod. Programarea orientata catre obiect le organizeaza altfel decit cea traditionala. Noua abordare pune o serie de limitari de acces asupra acestora, care conduc in final la o crestere a lizibilitatii codului pentru aplicatiile complexe.

Caracteristicile organizarii codului si datelor sint oglindite in sintaxa limbajelor orientate catre obiect (in engleza, Object Oriented Languages, OOL). Mai mult, deoarece exemplele sintactice extind unele lucruri deja cunoscute programatorilor, o privire asupra pseudocodului care urmeaza s-ar putea dovedi instructiva. Pseudocodul folosit in continuare nu este specific vreunui limbaj orientat catre obiect particular, dar este suficient de reprezentativ pentru acestea.

Acest articol isi propune, pentru inceput, sa treaca in revista avantajele pe care le confera programarea orientata catre obiect. In continuare prezentam o analiza paralela a modurilor conventional si orientat catre obiect de organizare a codului si datelor. Exemplele in pseudocod sint folosite pentru a introduce notiunile de: ***clasa***, ***incapsulare***, ***abstractizare a datelor*** si ***instantiere a obiectelor***. In final incercam sa prezentam felul in care avantajele amintite deriva din noua organizare a aplicatiilor in cadrul programarii orientate catre obiect.

Avantajele programarii orientate catre obiect

Scrierea aplicatiilor intr-un limbaj orientat catre obiect este mai usoara decit scrierea lor conventionala. Aplicatiile sint modele ale unor lucruri materiale sau situatii reale intilnite in mod obisnuit. Scopul aplicatiilor este de a simula asemenea lucruri. Intr-o analiza finala, aplicatia este judecata si considerata buna in masura in care este capabila de o asemenea simulare.

In cazuri concrete, inainte de a incepe activitatea de programare, este necesara o discutie cu beneficiarul. Pentru a-i putea indeplini cerintele trebuie sa intelegeti lumea obiectelor sale si comportamentul acestor obiecte. Incercati sa controlati logica problemei sale. De cele mai multe ori insa, exista o deosebire sintactica esentiala intre formalizarea logica a problemei si codul program asociat rezolvarii ei. Codul final intr-o aplicatie conventionala este atit de putin asemanator cu orice lucru cunoscut de catre un beneficiar mediu, incit pentru el acesta este total lipsit de semnificatie. Deoarece caile limbajelor conventionale sint atit de diferite de cele ale lumii reale, trebuie sa ne detasam de ceea ce trebuie modelat, tocmai pentru ca modelul sa ne reuseasca.

Intr-o alta forma de calcul, calculul analogic, exista un paralelism perfect intre sistemele modelatoare si cele modelate. Cantitatile modelate de catre un calculator analogic corespund caracteristicilor circuitelor hardware. De exemplu, tensiunea electrica dintre doua fire poate creste sau

scadea asemenea traiectoriei unui proiectil. Conectind cele doua fire la un plotter vom putea reproduce traiectoria pe hirtie. In acest caz elementul real si caracteristicile fizice ale calculatorului coincid. Componentele hardware se aseamana atit de mult cu modelul incit nu mai este nevoie de un software sofisticat. Intr-o asemenea situatie *programarea* se reduce la proiectarea hardware-ului.

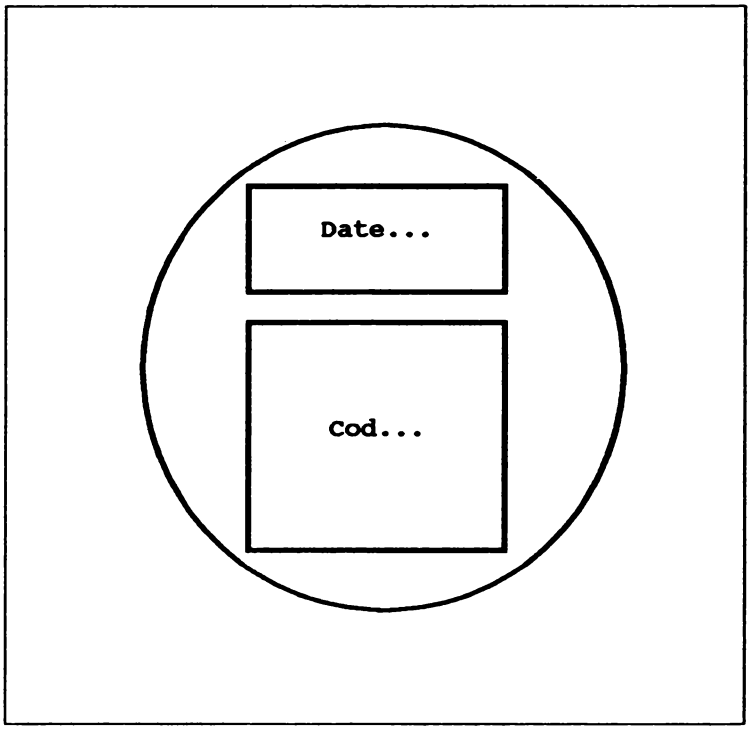


Figura 1. Un obiect

Undeva la mijloc se plaseaza si programarea orientata catre obiect. La fel ca in calculatoarele analogice, lucrul esential il constituie puternica analogie cu lumea reala. Dar in acest caz analogia este incorporata intr-o componenta software. Atingerea acestui deziderat inseamna construirea unor componente software care sa corespunda si sa se comporte la fel ca lucruri ale lumii reale. Codul si datele apartin limbajului la fel ca in programarea conventionala. Programarea orientata catre obiect le combina insa diferit. De fapt, insa, in programarea conventionala ele nu sint de loc combinate. Produsul final, "obiectul" logic, se comporta ca un obiect real. Mai mult, relatiile intre obiectele software reproduc relatiile existente intre obiectele reale. De aceea programele, ca sisteme logice alcatuite din obiecte logice, pot modela indeaproape sistemele reale alcatuite din obiecte reale. Unii dintre sustinatorii infocati ai programarii orientate catre obiect considera ca toate aplicatiile viitoare vor consta din "obiecte" standard stocate in biblioteci. Ei compara "obiectele" software cu chip-urile hardware, bibliotecile de obiecte standard cu cataloagele Intel si pe programatorii de aplicatii cu proiectantii placilor cu circuite integrate. Programarea orientata catre obiect are acelasi potential in ceea ce priveste standardizarea, refolosirea si productivitatea de care a beneficiat atit de mult hardware-ul. Vom vedea in continuare felul in care un limbaj conventional (limbajul C) isi stocheaza si manipuleaza datele. Apoi vom incerca sa punem in valoare alternativa programarii orientate catre obiect.

Organizarea codului si a datelor in programarea conventionala

Cum sint structurate codul si datele in programarea conventionala? Mai intii ne vom ocupa de organizarea codului. In limbajele conventionale, si limbajul C nu face exceptie, unitatile de cod ce intra in alcatuirea aplicatiilor sint *rutinele*. Orice aplicatie este vazuta ca o colectie de rutine.

Fie ca sint denumite *functii* in C, *subrutine* in FORTRAN, *proceduri* in dBASE sau Pascal, aceste rutine sint portiuni separate de cod apelabil, cu un singur punct de intrare si un singur punct de iesire. Rutinele transfera controlul inapoi partii de aplicatie care a initiat apelul. Structural, rutinele sint indiferente fata de codul care le apeleaza. Toate rutinele sint pe picior de egalitate si fiecare e libera sa apeleze oricare alta. Vom vedea in continuare felul in care programarea orientata catre obiect plaseaza o serie de restrictii in acest domeniu.

Ce putem spune despre date? Acestea sint mentinute in zone de memorie separate de cele care contin codul. Cum poate o rutina sa acceseze o anumita data? Simplu, cunoscind adresa exacta a acesteia. Adresa poate fi disponibila in mod public, ca in cazul variabilelor globale, sau primita ca parametru trimis printr-o stiva. Accesul la date este doar o problema de cunoastere a locatiei in care acestea se afla. De fapt, accesul la date este la fel de

deschis si nerestrictiv ca accesul la cod. Datele sint elementul pasiv iar codul, elementul activ. Orice rutina poate folosi orice data. Pe de alta parte, codul si datele sint separate. Rutinele manipuleaza datele "la distanta". Nu exista o asociere "intima" intre rutinele individuale si datele particulare. Titlul unei carti populare a lui Niklaus Wirth ilustreaza clar punctul de vedere conventional: "Algorithms + Data Structures = Programs". Pentru ca datele si codul sa poata fi combinate pentru a alcatui programe mai intii ele trebuiesc sa fie separate. Acesta este un alt aspect asupra caruia programarea orientata catre obiect isi plaseaza restrictiile.

Organizarea codului si datelor in programarea orientata catre obiect

Lucrul esential care trebuie inteles in programarea orientata catre obiect este ca obiectele sint combinatii impachetate de cod si de date, ca in figura 1. In al doilea rind se plaseaza detaliile de impachetare iar in al treilea rind, comportamentul rezultat. Ambele obiecte, sintactice si operationale sint pachete de cod si date legate impreuna.

Aspectele sintactice se vor reflecta in codul care defineste obiectele in limbajele orientate catre obiect. Acesta implica adeseori unele declaratii subordonate datelor si codului constituent. Operational, obiectele se vor "insufleti" in cursul executiei unui program. In acel moment, codul si datele declarate, facind parte din acelasi obiect, vor poseda asociatii structurale si operationale necunoscute in modelul conventional. Pe scurt, codul si datele unui obiect sint limitate unul altuia dupa cum urmeaza. Codul din afara obiectului nu poate accesa datele acestuia, si, reciproc codul intern obiectului nu poate accesa datele din exterior. Programul ca intreg se reduce la o colectie de asemenea obiecte impachetate, ca in figura 2.

Compunerea obiectelor

In limbajele orientate catre obiect (OOL) spunem despre definitia obiectelor ca este "hibrida", deoarece o asemenea definitie mixeaza declaratiile de cod cu cele de date. Codul din afara unui obiect are acces la acesta doar prin intermediul apelurilor la rutinele proprii obiectului. Acestea devin partile autorizate pentru accesul la date. Rutinele pot manipula date, pot apela rutine in alte obiecte, si/sau intoarce informatie. Limbajele orientate catre obiect (OOL) au inglobate mecanisme de control al violarii de acces. Asemenea mecanisme exista si in limbajele

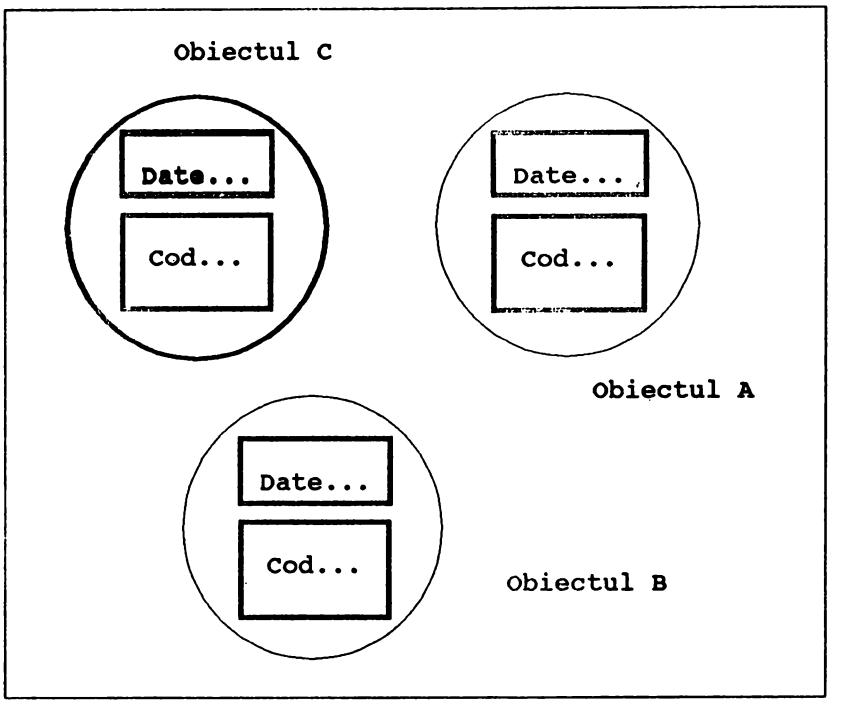


Figura 2. Organizarea codului si datelor in OOP

conventionale C, dBASE, etc, dar ele sint mult mai elaborate in limbajele orientate catre obiect. Figura 3 ilustreaza aceste concepte.

In modul clasic, procedural, codul "poseda" anumite date. In noua abordare datele, la rindul lor, poseda codul. Aceasta confera datelor un comportament caracteristic. In mod conventional datele erau "pasive". In dBASE de exemplu, ele asteapta sa fie manipulate. Programatorii care folosesc limbajele orientate catre obiect tind sa ajusteze aceasta conceptie.

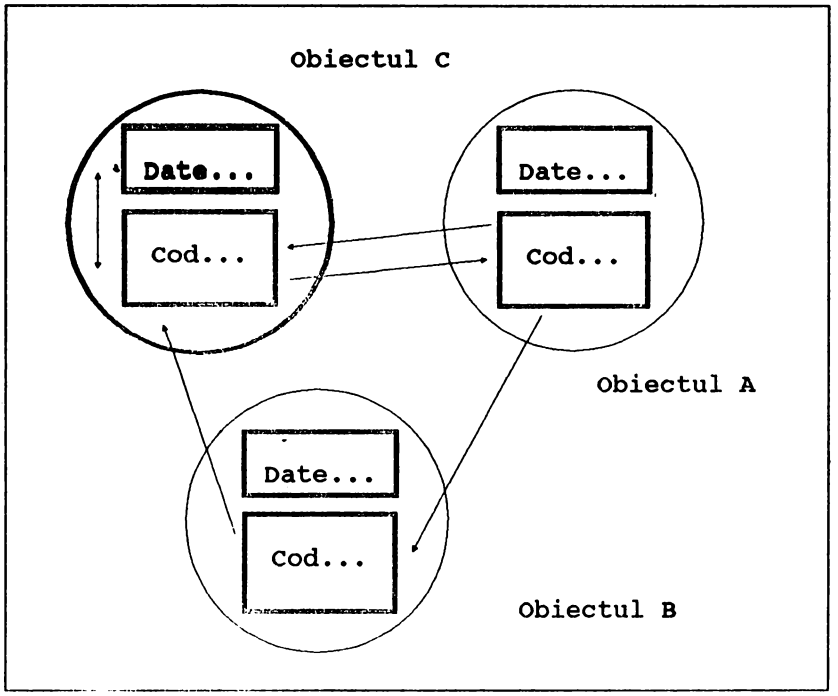


Figura 3. Recursia unei aplicatii OOP

```
     end declare
end define
```

De notat ca datele iau forma *variabilelor de instantiere,* iar codul apare sub forma *metodelor.* Intreg procedeul de impachetare este denumit *impachetare* (in engleza, *encapsulation*).

De la clase la obiecte

Simpla definire a unei clase nu creaza nici un numarator. Pentru a crea un obiect de acest tip, in functie de limbajul orientat catre obiect folosit, va trebui sa emiteti o comanda aditionala de forma:

```
     unit_count :=
counter:new()
     lei_count  :=
counter:new()
```

Variabile de instantiere + Metode = Clase

Vom introduce o parte din terminologia specifica programarii orientate catre obiect printr-un exemplu, folosind o definitie concreta a unui obiect. Tipurile de obiecte sint denumite *clase*, iar pseudocodul asociat se numeste definitia clasei. Iata definitia clasei pentru un obiect practic folosit in programare, un numarator (counter):

```
define class "counter"
     declare instance variables
          an integer called "value"
     end declare
     declare methods
          method initialize_to_zero
               value := 0
          end method
          method increment
               value := value + 1
          end method
          method decrement
          value := value - 1
          end method
          method access_value
               return (value)
          end method
```

In acest moment unit_count si lei_count devin instante ale unui numarator. Instantierea unei clase poarta denumirea de *obiect*. In momentul instantierii unui obiect, variabilele de instantiere declarate in definitia clasei sint alocate in memorie. In cazul nostru un singur intreg denumit "valoare" va fi alocat pentru un numarator. Cele doua variabile de instantiere vor fi separate si distincte, diferentierea dintre ele va fi facuta prin intermediul unor referinte de forma:

```
unit_count:access_value()
lei_count:access_value()
```

pentru a indica "valoarea" carui obiect este ceruta. Aceste variabile, dupa cum s-a explicat, pot fi accesate doar de catre unitatea de cod care apartine aceluiasi obiect, adica una din cele patru metode care apartin numaratorului. Spre deosebire de variabilele de instantiere, nu se va incarca memoria cu mai multe copii ale metodelor pentru fiecare obiect. Cel mult o copie a unei rutine va fi incarcata pentru o clasa, aceasta fiind disponibila pentru toate instantele sale. Dar acesta este deja un detaliu de implementare.

Conversatia cu obiectele. Mesajele.

Cum folosesc programele un obiect? Simplu, trimitindu-i *mesaje*. Aceasta se realizeaza prin apelarea uneia din metodele specifice obiectului. In exemplul nostru daca vindem o unitate de produs pentru 2 lei, va trebui sa cfestem numaratorul unitatilor cu 1 si pe cel al leilor cu 2. Nu putem ataca problema direct:

```
unit_count.value =
    unit_count.value + 1
lei_count.value  =
        lei_count.value + 2
```

deoarece codul nostru nu are acces la aceste variabile. Este necesar sa apelam la metodele ce apartin acestor obiecte. Sintaxa ar putea arata astfel:

```
unit_count:increment()
lei_count:increment():increment()
```

Privind dintr-o perspectiva procedurala asistam la apelarea functiei "increment" aplicata o data in unit_count si de doua ori valorii din lei_count. Intr-o perspectiva orientata catre obiect noi apelam unit_count pentru a se autoincrementa, comunicindu-i mesajul corespunzator. Intr-o asemenea abordare sintem liberi sa ignoram mijloacele interne, de realizare a comenzii, invizibile si indiferente noua, si sa ne concentram asupra facilitatilor acestei comenzi.

De notat ca nu putem face nimic in plus cu obiectul in afara de ceea ce el poate realiza prin propriile metode. De exemplu, nu putem aduna direct 2 lei la valoarea din lei_count deoarece nu exista rutine pentru asa ceva printre metodele din definitia clasei. Acest lucru poate fi totusi realizat prin trimiterea de doua ori a mesajului de incrementare. Aceasta restrictie confera doua mari beneficii metodei. Mai intii ca este imposibila modificarea neintentionata a valorii lei_count in cadrul aplicatiei. Aceasta confera programarii orientate catre obiect *fiabilitate*. In al doilea rind, deoarece accesul la obiectul lei_count este centralizat, orice schimbare in codul central se va propaga in intreaga aplicatie. In cazul in care se modifica metoda mesajele ramin neschimbate. Aceasta face ca obiectele sa fie *usor de intretinut*. Similitudinea cu chip-urile hardware, de care aminteam mai sus, este perfect ilustrata de exemplul prezentat.

Specializarea obiectelor. Ierarhia claselor si mostenirea.

O aplicatie reala, asemenea lumii pe care o modeleaza, va contine mai multe tipuri de obiecte, cu multe asemanari si deosebiri intre ele. Programarea orientata catre obiect exprima aceasta in felul in care puteti deriva un tip de obiect din altul. La definirea unei noi clase, puteti stipula identitatea, pina la o diferenta specifica, cu o clasa deja existenta. Aceasta poarta numele de *mostenire* si conduce la o *ierarhie de clase*.

"Programarea diferentiala" sau "programarea prin modificare" este caracteristica programarii orientate catre obiect.

In continuare vom construi un numarator specializat, de o singura cifra, care sa mearga de la 0 la 9. Programarea orientata catre obiect va permite sa-l definiti incremental astfel:

```
define class "1-digit counter",
     inherit from "counter"
     declare methods
     method increment
          if value = 9
               value := 0
          else
               value := value + 1
          end if
     end method
     method decrement
          if value = 0
               value := 9
          else
               value := value - 1
          end if
     end method
     end declare
end define
```

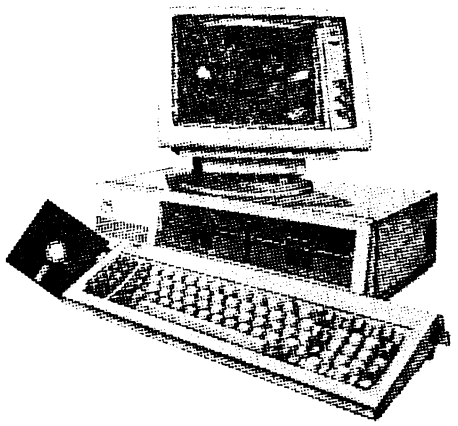
Cele doua metode explicitate in subclasa "1-digit counter" se suprapun peste definitia clasei parinte. Ele inlocuiesc metodele parinte cu acelasi nume, in cazul in care acestea exista. Altfel, metoda parinte (asemenea datelor in multe implementari) devine automat parte a definitiei subclasei. Metoda fiului "mascheaza" metoda parintelui. Cazul inlocuirii este denumit uneori "umbrire".

Clasele presupun o organizare ierarhica, cu o mare generalitate la nivelele superioare si o mare specializare la cele inferioare. Folosind mostenirea in cadrul unei ierarhii pentru a defini

comportamentul obiectelor, evitam redundanta in definirea aceluiasi comportament pentru fiecare obiect in parte.

Organizarea aplicatiilor in programarea orientata catre obiect

Reorganizarea componentelor de limbaj in programarea orientata catre obiect conduce in mod natural la o reorganizare a aplicatiilor. La nivelul designer-ului de aplicatii, obiectele sint noile unitati atomice de compunere a software-ului.



Programatorii pot fi acum designeri de obiecte, designeri de aplicatii sau ambele. Construirea obiectelor implica o mai mare detaliere asemenatoare programarii normale intr-un limbaj procedural. Deoarece construirea aplicatiilor inseamna folosirea unor obiecte deja testate, aceasta reprezinta un nivel mai evoluat al programarii.

Definirea obiectelor pe care le folositi (de fapt, clase) poate fi furnizata de catre autorii limbajului. Limbajele orientate catre obiect tipice sint livrate ca o mare colectie de clase (de exemplu Microsoft Windows "Software Development Kit") si nu ca o biblioteca de functii.

Odata ce obiectele au fost alcatuite, le veti folosi ca o cutie neagra. Va trebui sa evitati sa mai ginditi in termeni de cod si date, ci va trebui sa va concentrati asupra mesajelor lansate obiectelor, cerindu-le sa realizeze actiuni asupra lor insele.

Rezolvarea orientata catre obiect a problemelor.

Obiectele pe care le asamblati in aplicatiile dumneavoastra sint alese in mod deliberat. Am spus mai devreme ca obiectele software reflecta obiectele reale si relatiile dintre ele. Cautati sa construiti sisteme de aplicatii care sa aproximeze cit mai bine portiunea de realitate pe care doriti sa o modelati. In acest fel, specificarea problemei de catre beneficiar va dura mai putin.

Un asemenea procedeu este denumit *rezolvarea orientata catre obiect a problemelor*. Aceasta consta din patru pasi:

- specificarea problemei;
- identificarea obiectelor intr-o solutie;
- identificarea mesajelor la care obiectele trebuie sa raspunda;
- stabilirea unei secvente de mesaje catre obiectele care vor furniza o solutie problemei puse.

Construirea obiectelor are loc in pasii 2 si 3, construirea aplicatiei in pasul 4.

Limbajele orientate catre obiect sint uneltele care fac posibila programarea orientata catre obiect. Avantajele fata de programarea procedurala apar in domeniul intretinerii, refolosirii si fiabilitatii aplicatiilor. Aceste beneficii sint datorate noii forme de agregare si minuire a codului si datelor, obiectul. Citeva caracteristici ale obiectelor (incapsularea, mostenirea, polimorfismul) permit realizarea unui paralelism structural si comportamental intre componentele software si componentele lumii reale pe care o simuleaza, lucru imposibil de realizat in limbajele procedurale.

Programarea orientata catre obiect se constituie intr-una din cele mai promitatoare tehnologii ale ingineriei software moderne.

Bibliografie:

- "Concepts of OOP", David Morgan, Nantucket News, Volume 4, Number 1, 1989;
- "What is Object Oriented Programming", Addison Wesley Publishing Company, 1988.

Ce nu prea se stie despre calculatoarele SPECTRUM

Mircea Crutescu

Fisiere

Printre multele trucuri posibile pe calculatoarele compatibile Spectrum, se gasesc si rutinele de acces la fisierele inregistrate pe casete. Inainte de a scrie aceste mici rutine, intotdeauna ar fi trebuit folosit BASIC-ul pentru a incarca fisiere. In cazul aplicatiilor unde se doreste utilizarea tehnicilor de "windowing", la citirea unui fisier, prin folosirea comenzii "LOAD", sistemul afiseaza textul "Program:xxxxx" sau "Bytes:yyyyy", netinind cont (neavind cum) de ferestrele create.

Toate aceste inconveniente sint depasite folosind citeva rutine simple de incarcare separata a blocului "header" si a blocului de date. A doua rutina poate fi folosita si pentru a incarca asa numitele blocuri "headerless".

Un header standard de fisier pe caseta are 17 octeti lungime, utilizati dupa cum urmeaza (folosind notatie hexa):

- 00: codificare fisier (0, 1, 2, 3 - vezi mai jos);
- 01-0A: nume fisier (daca in octetul 01 se gaseste valoarea FF, atunci numele se considera nul);
- 0B-0C: lungimea fisierului (a blocului de date);
- 0D-10: parametri functie de tipul fisierului.

Codificarea tipurilor de fisiere din octetul 00 este facuta astfel:

- 0 inseamna că blocul ce urmeaza este un program sursa BASIC si in acest caz, octetii 0D-0E contin linia de autolansare a programului (codificata ca un intreg, adica cel mai putin semnificativ octet este memorat in 0D), iar octetii 0F-10 contin adresa de memorie unde se rezerva spatiu pentru variabilele programului;
- 1 si 2 sint codurile pentru "Number array" si "Character array", respectiv;
- 3 este codul pentru un bloc de octeti care poate contine orice; un astfel de cod este pus de sistem in header in cazul in care se executa comanda "SAVE "nume" CODE adresa_inceput, lungime".

In octetii 0D-0E este stocata adresa de inceput a blocului, din momentul salvarii lui pe caseta. Dupa cum stiti, probabil, aceasta adresa este folosita de sistem in cazul in care la o comanda "LOAD "nume" CODE" nu specificam unde dorim sa incarcam blocul. Octetii 0F-0A contin lungimea blocului.

Cu ajutorul urmatoarei rutine in limbaj de asamblare, header-ul unui fisier poate fi incarcat oriunde se doreste, pentru a putea fi decodificat ulterior:

```
ZECIMAL             ASAMBLARE
055                 SCF             ; seteaza carry
062 000             LD A,0
221 033 ??? ???     LD IX,NN        ; NN = adresa de inceput
017 017 000         LD DE,17        ; 17 = lungimea headerului in octeti
205 086 005         CALL 1366       ; apel rutina de incarcare din ROM
201                 RET
```

Printr-o modificare minora a acestei rutine se obtine cea de incarcare a blocului urmator headerului, despre care acum exista toate informatiile.

```
ZECIMAL              ASAMBLARE
055                  SCF          ; seteaza carry
062 255              LD A,255     ; modificarea !!!
221 033 ??? ???      LD IX,NN     ; NN = adresa de inceput
017 017 ??? ???      LD DE,MM     ; MM = lung. headerului in octeti
205 086 005          CALL 1366    ; apel rutina de incarcare din ROM
201.                 RET
```

Pentru a asigura protectia fisierelor dumneavoastra puteti salva fisiere "headerless" folosind:

```
ZECIMAL              ASAMBLARE
167                  AND A        ; sterge carry
062 255              LD A,255
221 033 ??? ???      LD IX,NN     ; NN = adresa de inceput
017 ??? ???          LD DE,MM     ; MM = lung. headerului in octeti
205 194 004          CALL 1218    ; apel rutina de salvare din ROM
201                  RET
```

Valorile in zecimal sint date pentru cei care nu poseda un asamblor Z80, acestia putind totusi folosi rutinele de mai sus, introducindu-le octet cu octet in memorie, cu ajutorul instructiunii "POKE".

Salturi

Un alt truc putin cunoscut este posibilitatea de a face un salt la o anumita linie a programului, la o anumita instructiune (instructiunile fiind delimitate pe aceeasi linie cu ":"). Cu trei "POKE"-uri problema este rezolvata:

```
POKE 23618,cel mai putin semnificativ octet al numarului liniei;
POKE 23619,cel mai semnificativ octet al numarului liniei;
POKE 23620,numarul instructiunii de pe linie.
```

Protectie

Pentru obtinerea unui text program total nelizibil, incercati:

```
POKE 23607,50 (sau alta valoare)
```

Pentru revenirea la normal:

```
POKE 23606,0
POKE 23607,60
```

In numarul viitor, citeva trucuri despre grafica si transferul parametrilor catre rutine scrise in limbaj de asamblare Z80 prin mecanismul functiilor definite cu FN(). De asemenea, redactia pune la dispozitia celor interesati documentatie pentru utilitare SPECTRUM.

A simula? Nimic mai simplu...

Continuam serialul nostru inceput in numarul precedent printr-o invitatie la o "Licitatie cu obiecte de arta". Iata scenariul pe care vi-l propunem. In aceasta simulare, veti avea ocazia sa cumparati si sa vindeti cel mult 5 tablouri. Obiectivul jocului este de a obtine cel mai mare profit cumparind tablourile cit mai ieftin posibil si vinzindu-le apoi cit mai scump.

Pentru a cumpara un tablou, va trebui sa licitati impotriva ofertei secrete a unui alt cumparator (calculatorul). Cind un tablou este oferit spre vinzare, vor fi prezentate 3 numere care reprezinta media si intervalul in care sint cuprinse ofertele pentru acesta. De exemplu, "200 300 400" indica faptul ca oferta medie pentru aceea pictura este de 300, iar aproximativ 70% din timp pretul se va situa intre 200 si 400. De notat ca tablourile cele mai scumpe tind sa aiba un interval mai mare de variatie a preturilor.

Dupa ce veti achizitiona picturile, veti avea ocazia sa le vindeti. In acest scop veti primi intre una si cinci oferte, fara a sti in avans numarul exact al acestora. Ofertele vor fi, in medie, cu 50 mai mari decit ofertele facute in faza de cumparare. In cazul in care nu acceptati oferta, si aceasta este ultima care vi se face, aceasta va fi acceptata automat. Uneori va fi mai bine sa acceptati un pret mai mic decit pretul de achizitie decit sa asteptati o oferta care s-ar putea sa nu se materializeze niciodata.

In momentul in care toate picturile pe care le-ati cumparat vor fi vindute, vi se va afisa profitul total pentru intreaga tranzactie.

Exemplu de rulare:

```
CUMPARA PICTURA 1
PRETURI:  546  553  560
OFERTA TA?  560
OFERTA ADVERSAR  565.
AI PIERDUT.

CUMPARA PICTURA 2
PRETURI:  336  449  562
OFERTA TA?  400
OFERTA ADVERSAR  440.
AI PIERDUT.

CUMPARA PICTURA 3
PRETURI:  213  288  363
OFETA TA?  300
```

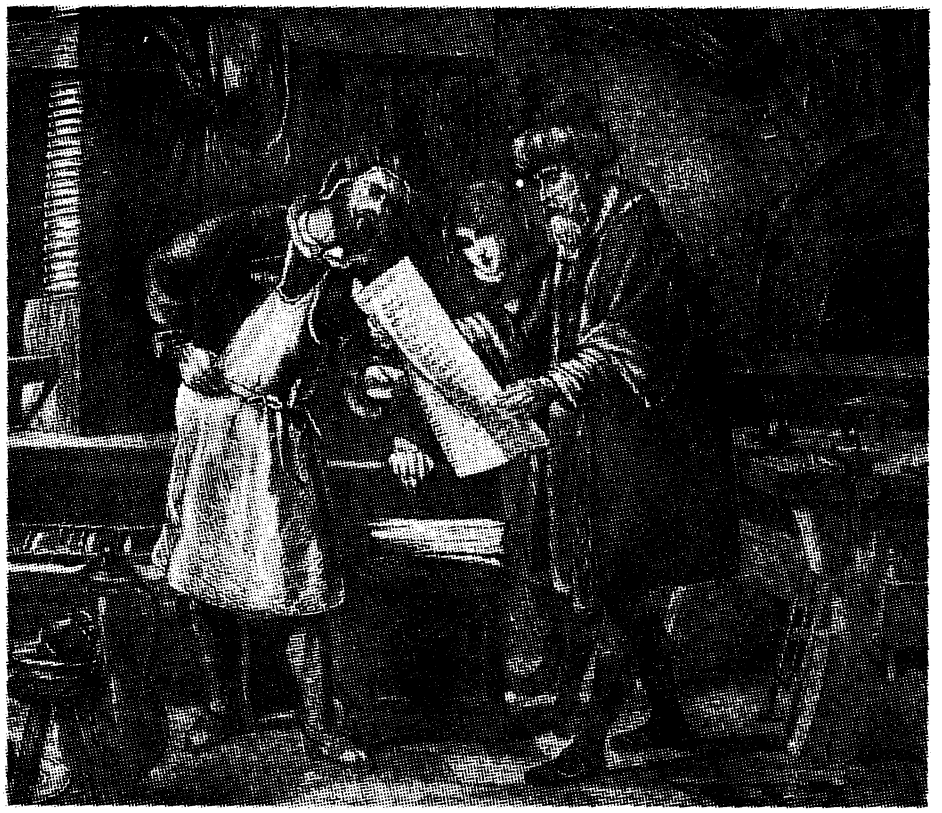
Listing: ARTS.BAS

```
010  REM SETARE PRETURI SI DOMENII
020  DIM P(5),S(5),F(5)
030  FOR I=1 TO 5
040  P(I)=100+INT(900*RND(1))
050  S(I)=INT(P(I)*RND(1))
060  IF P(I) > 500 THEN S(I)=INT(P(I)*.7*RND(1))
070  F(I)=0
080  NEXT I

090  REM CUMPARARE TABLOURI
100  FOR I=1 TO 5
110  GO SUB 500
120  PRINT: PRINT "CUMPARA PICTURA"; I:PRINT:PRINT
130  PRINT "PRETURI:"; INT(P(I)-.5*S(I)); P(I);
     INT(P(I)+.5*S(I))
140  PRINT: PRINT: INPUT "OFERTA TA"; OT
150  PRINT "OFERTA ADVERSAR"; OC; "."
160  IF OT > OC THEN PRINT "AI CUMPARAT-O."; F(I)=OT;
```

```
   GO TO 180
170  PRINT "AI PIERDUT.".
180  NEXT I

195  REM VINZARE TABLOURI
200  FOR I=1 TO 5
210  IF F(I)=0 THEN 310
220  FOR K=1 TO INT(5*RND(1))
230  GO SUB 500: OC=OC+INT(100*RND(1))

240  PRINT "VINDE PICTURA"; I
250  PRINT "AI CUMPARAT-O PENTRU"; F(I): PRINT "OFERTA
   MEDIE
     ESTE"; P(I)+50
260  PRINT "OFERTA"; K; "ESTE"; OC; "."
270  INPUT "ACCEPTI"; D$
280  IF D$="D" THEN 300
290  NEXT K
300  P=P+OC-F(I)
310  NEXT I
320  PRINT: PRINT "PROFITUL TAU ESTE"; P; "."
330  INPUT "MAI JOCI"; D$
340  IF D$="D" THEN RUN
350  END

495  REM SUBRUTINA DISTRIBUTIA NORMALA
500  D=0
510  N=INT(65536*RND(1))
520  FOR J=1 TO 16
530  Q=INT(N/2)
540  D=D+2*(N/2-Q)
550  N=Q
```

Listing: ARTS.BAS (continuare)

```
OFERTA ADVERSAR  324.
AI PIERDUT.

CUMPARA PICTURA 4
PRETURI:  403  514  625
OFERTA TA?  600
OFERTA ADVERSAR  497.
AI CUMPARAT-O.

CUMPARA PICTURA 5
PRETURI:  274  346  417
OFERTA TA?  350
OFERTA ADVERSAR  311.
AI CUMPARAT-O.

VINDE PICTURA 4
AI CUMPARAT-O PENTRU 600.
OFERTA MEDIE ESTE 564.
OFERTA 1 ESTE 649.
ACCEPTI? D

VINDE PICTURA 5
AI CUMPARAT-O PENTRU 350.
OFERTA MEDIE ESTE 396.
OFERTA 1 ESTE 365.
ACCEPTI? N
```



```
PROFITUL TAU ESTE 64.
MAI JOCI?
```

Variabile

```
P(5)    Preturi.
S(5)    Domeniul preturilor.
F(5)    Indicator setat daca
          tabloul a fost vindut.
OC      Oferta adversar.
OT      Oferta ta.
I,J,K   Indici.
P       Profit.
N       Numar.
D       Deimpartit.
Q       Cit.
```

Modificari posibile

Modificari minore

1. Numarul picturilor - liniile 20, 30, 100, 200;

2. Preturile de start - linia 40;

3. Profitul intern - liniile 230, 250;

4. Numarul ofertelor - linia 220;

Modificari majore

1. Introduceti, ca element de suspans, spargerea. Ea poate aduce jucatorului prejudicii asupra profitului final.

2. Presupuneti ca unele dintre picturile achizitionate foarte ieftin sint de fapt niste capodopere.

3. Modificati programul astfel incit sa puteti juca simultan cu mai multi adversari.

Protejarea discului fix impotriva virusilor

Avantajul discurilor fixe fata de discurile flexibile este indiscutabil din punctul de vedere al vitezei de acces si al capacitatii de memorare. Singurul dezavantaj este doar imposibilitatea decuplarii acestei mari magazii de informatii. Nu se poate scoate discul din unitate, nici macar nu se poate proteja contra scrierii accidentale sau rau intentionate pe el. Decuplarea fizica s-ar putea realiza prin modificari hardware, dar nu este recomandabila. In schimb se poate realiza protejarea logica de tip "Read/Only" a discului fix cu ajutorul facilitatii sistemului de operare, facilitate pe care se bazeaza si functionarea virusurilor informatice. Pentru cei curiosi prezentam in listingul alaturat o varianta de astfel de program, pe care l-am scris in limbaj de asamblare si l-am denumit WINCHRO (de la "WINCHester Read Only"). Din programul sursa prezentat se obtine programul executabil prin urmatoarea secventa de comenzi DOS:

```
MASM winchro
LINK winchro
EXE2BIN winchro.exe winchro.com
DEL winchro.exe
```

Mesajul de avertisment emanat de catre LINK este absolut normal si, in consecinta, trebuie pur si simplu ignorat. Programul rezultat, WINCHRO.COM, are o lungime de numai 96 de octeti.

Pogramul, odata lansat, modifica adresa rutinei de tratare a intreruperilor de scriere/citire pe/de pe disc (intreruperea BIOS 13H), salvind in prealabil adresa rutinei vechi. Dupa aceea prima parte a programului ramine rezidenta in memorie (0.2 Kocteti). Aceasta parte se activeaza ulterior ori de cite ori se incearca un acces pe un disc oarecare (via INT 13H). Daca tipul accesului este scriere (codul functiei, registrul AH = 03H) pe un disc fix (identificatorul discului, registrul DL = >80H),

Listing: WINCHRO.ASM

```
SegBIOS   SEGMENT  AT   40H       ; Segment BIOS     ;
     ORG  74H                    ;                  ;
StDisc    DB   ?                 ; stare disc       ;
SegBIOS   ENDS                   ;==================;
SegCod    SEGMENT PARA PUBLIC 'CODE'   ; Segment de cod   ;
     ORG  0100H                  ; Pentru programe COM   ;
     ASSUME    cs:SegCod,ds:NOTHING    ;                ;
Start:    push cs                ;                  ;
     pop  ds                ; ds:=cs           ;
     ASSUME    ds:SegCod         ;                ;
     jmp  Ins               ;                  ;
Rutina    PROC FAR               ;==================;
     ASSUME    ds:NOTHING          ; Rutina noua de disc   ;
     sti                    ;==================;
     cmp  dl,80H                 ; DL=>80H daca disc fix ;
     jb   Retur                  ;                  ;
     cmp  ah,03H                 ; 03H=write        ;
     je   Eroare                 ;                  ;
Retur:    jmp  DWORD PTR cs:[AdrBIOS]  ; Salt la rutina veche  ;
Eroare:   push ax                ;==================;
     pushf                       ;                  ;
```

Listing: WINCHRO.ASM (continuare)

```
     push ds             ; Eroare:      ;
     mov  ax,40H            ;=======================;
     mov  ds,ax             ;                       ;
     ASSUME    ds:Seg3IOS        ; Setare Carry      ;
     mov  al,80H            ;                       ;
     mov  StDisc,al      ; pentru eroare          ;
     pop  ds             ;                       ;
     popf                ; Drive Not Ready    ;
     pop  ax             ;                       ;
     ASSUME    ds:NOTHING        ; pentru protejarea discului ;
     mov  ah,80H            ; fix impotriva scrierii     ;
     stc                 ;                       ;
     ret  2              ;                       ;
Rutina    ENDP             ;===============================;
AdrBIOS   DW   2 DUP (?)          ; Adresa rutinei BIOS        ;
Ins: ASSUME    ds:SegCod     ;===============================;
     mov  ax,3513H       ; Instalarea in memorie:     ;
     int  21H            ; citirea adresei vechi      ;
     cmp  bx,OFFSET Rutina   ;                    ;
     jne  NeIns             ; daca este deja instalat    ;
     mov  ax,4C00H       ;                       ;
     int  21H            ; atunci stop.           ;
NeIns:    mov  WORD PTR AdrBIOS,bx     ; Neinstalat:           ;
     mov  WORD PTR AdrBIOS+2,es   ; memorare adresa veche      ;
     mov  dx,OFFSET Rutina   ;                    ;
     mov  ax,2513H       ; modificare adresa noua     ;
     int  21H            ;                       ;
     mov  ax,OFFSET Ins      ; se lasa programul          ;
     mov  cl,04H             ; rezident in memorie        ;
     add  ax,0FH             ; pina la adresa Ins         ;
     shr  ax,cl              ;                       ;
     mov  dx,ax              ;                       ;
     mov  ax,3100H           ;                       ;
     int  21H                ; se termina programul       ;
```

rutina returneaza un cod de eroare de tip "Not Ready". In caz contrar transfera controlul rutinei anterioare de tratare a intreruperii respective.

Asa cum este conceput, programul ramine inefectiv daca se mai lanseaza inca o data dupa ce a fost deja instalat in memorie. Revenirea la starea initiala de "Read/Write" a discului fix nu se poate realiza decit prin reinitializarea sistemului, prin apasarea butonului RESET sau a combinatiei de taste `Ctrl+Alt+Del`.

Este util sa se lanseze acest program inainte de executarea unui program care este susceptibil de a fi fost contaminat cu un virus. In general aceste programe se lanseaza de pe discuri flexibile si nu ar trebui sa *scrie* pe discul fix (uneori este normal sa *citeasca* de pe acesta, de pilda, informatii despre sistemul de operare). Daca vor incerca totusi sa scrie, atunci va apare mesajul de eroare "Not ready error writing drive X:", unde X este discul pe care s-a incercat accesul (C, D, ...), lucru care trebuie sa dea de gindit in ceea ce priveste starea de "sanatate" a programului testat.

PC-CLIP

Incepind cu acest numar, revista va contine o rubrica dedicata tuturor programatorilor care doresc sa impartaseasca si altora din experienta lor.

Ideea de la care am plecat, rezultata din experienta indelungata si nefericita, este aceea ca este pacat ca fiecare programator sa reinventeze roata la fiecare noua aplicatie pe care o scrie. Multi dintre noi isi irosesc timpul rezolvind ad-hoc o problema minuscula, in loc sa se foloseasca experienta altora, care s-au lovit de aceeasi problema. Ce rost are sa rescriem functii de zeci de linii precum `strrev()`, ca sa luam un exemplu extrem, si sa ne batem capul cu intrebari de genul "`for i := 0 to n` sau `for i := 1 to n` ?!", in loc sa folosim o biblioteca de functii si sa ne petrecem timpul concentrindu-ne asupra problemelor reale, de substanta?! Programarea, desi este o arta si nu o meserie, cum bine stiu toti cei care o practica, are totusi nevoie de instrumente adecvate pentru a fi eficienta cu adevarat.

Programele, functiile sau subrutinele nu trebuie sa depaseasca 80-120 de linii sursa, desigur, in functie de limbajul folosit. Se accepta orice limbaj, inclusiv fisiere de comenzi, si ADA daca doriti dumneavoastra.

Nu vom publica decit programe testate. Specialistii nostri vor testa fiecare program dintre cele selectate pentru publicare si isi vor asuma raspunderea pentru buna functionare a acestora.

Programele primite, dar nepublicate, nu vor fi restituite autorilor, ele raminind la sediul redactiei la dispozitia tuturor celor interesati. Ele vor fi disponibile pe dischete, iar periodic vom publica o lista completa a acestora.

Dorim sa realizam o biblioteca de programe, utilitare, functii, sau simple idei interesante, ceva in genul fondului imens de astfel de obiecte care exista pe sistemele UNIX. Speram ca in curind va fi data in exploatare si o retea nationala de calculatoare, lucru care va usura mult schimbul rapid si eficient de informatii intre specialisti dar nu numai intre acestia.

Ca sa spargem gheata, in acest numar prezentam un program, fara pretentia ca este ceva nemaipomenit, dar care se incadreaza bine in ceea ce spuneam mai sus despre rezolvarea ad-hoc a problemelor de fiecare zi: l-am scris in timp ce pregateam revista pentru tipar, pentru ca aveam nevoie, pur si simplu. Nu avem mai nimic de comentat despre el. Versiunea prezentata este compilata cu Borland Turbo C Versiunea 2.00, dar se poate adapta in doua minute pentru orice compilator MS-DOS.

Listing: SWEEP.C

```
/*
     sweep.c
     PC-Magazin 3/1990

     Executa recursiv o comanda MS-DOS in fiecare din
     sub-directoarele dominate de directorul curent.
     Utilizare: SWEEP comanda

     Exemple:
     ---------

     Scrie un fisier cu lista tuturor directoarelor
```

Listing: SWEEP.C (continuare)

```
     si fisierelor de pe disc:
          CD \
          SWEEP dir*>nume-fisier
     Colecteaza toate fisierele sursa de pe un disc, pe alt disc:
          CD \
          SWEEP copy *.c a: /v
     Sterge fisiere inutile:
          CD \
          SWEEP del *.bak
     Si asa mai departe...
*/
#include <dir.h>
#include <dos.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

char command [256];     /* memoreaza argumentele functiei main() */

/*************************************************************************/
void process (char *dir) {

     /*   Afiseaza ceva asemanator cu o linie de comanda
      *   MS-DOS (d:\path\command), apoi executa <command>,
      *   dintr-o variabila globala.
      *   Pentru succesul functiei system(),
      *   COMMAND.COM trebuie sa fie accesibil!
      */

     char *p;

     p = dir + strlen (dir) - 1;
     if ('\\' == *p)
          *p = 0;        /* elimina problema D:\\ */
     printf ("%s\\%s\n", dir, command);
     if (0 != command [0])
          system (command);
}

/*************************************************************************/
void sweep (char *dir, void (*func)(char *)) {

     /*   Functia recursiva de parcurgere a subarborelui de directoare.
      *   Apeleaza <func> pentru fiecare director din subarborele
      *   dominat de <dir>, inclusiv pentru acesta, trimitindu-i ca
      *   parametru numele complet al sau.
      */

     register f;
     char save [MAXPATH];
     struct ffblk ff;
```

Listing: SWEEP.C (continuare)

```
                                                                    */
     getcwd (save, sizeof(save));     /* salveaza director curent DOS */
     chdir  (dir);                    /* trece in directorul de lucru */
     (*func)(getcwd (NULL, MAXPATH)); /* intoarce nume COMPLET */

     /* pentru findfirst() si findnext(), v. documentatie compilator */
     for (f=findfirst ("*.*", &ff, FA_DIREC); 0==f; f=findnext (&ff)) {
          /* daca este un sub-director autentic */
          if ((ff.ff_attrib & FA_DIREC) && '.' != ff.ff_name[0])
               /* atunci apeleaza recursiv pentru acel sub-director */
               sweep (ff.ff_name, func);
     }
     chdir (save); /* restaureaza starea */
}

/*********************************************************************/
void main (int argc, char *argv[]) {

     /*   main() colecteaza argumentele din
      *   linia de comanda, apoi preda controlul
      *   functiei recursive cu pricina.
      */

     register k;

     for (k=1; k < argc; k++) {
          strcat (command, argv[k]);
          strcat (command, " ");
     }
     /* apeleaza sweep() incepind din directorul curent, inclusiv! */
     sweep (getcwd (NULL, MAXPATH), process);
```

Preturi orientative la produse hardware/software pe piata internationala

Spicuiri din PC-Magazine, BYTE si PC-Week

IBM PS/2

Model 25-Mono/Color	$995/1250
Model 30-286 20 Mb Disk	$1795
Model 30-286 30 Mb Disk	$1925
Model 55SX-30 Mb Disk	$2795
Model 55SX-60 Mb Disk	$3050
Model 60-44 Mb Disk	$3200
Model 70-E61/061/A61	$3550, 3999, 5900
Model 70-120 Mb Disk 20 MHz	$4499
Model 70-A21-120 Mb Disk 25 MHz	$6300
Model 80-70 Mb Disk	$4800
Model 80-115 Mb Disk	$5900

COMPAQ

286e Deskpro 1/20/40	$1910/2210/2500
386/20e Deskpro 40/110 Mb	$4295/4875
386S Deskpro 20 Mb/40 Mb	$Call/2850
386/33 84/320 Mb	$7295/10599
386/25 110 Mb/300 Mb	$6200/8150
Portable III Model 20	$3400
Portable III Model 40	$4050
Portable 386 Model 40	$5200
Portable 386 Model 110	$6800
Laptop SLT 286 20/40	$3675/4175
Laptop LTe 286 20/40	$3229/3799

TOSHIBA LAPTOPS

T1000SE	$1179
T1200FB/HB	$1485/1879
T1600 20/40 Mb Disk	$3200/3600
T3100 SX	$3989
T3200 40Mb Disk	$3400
T3200 SX	$4000
T5200 40 Mb/100 Mb	$4889/5250

ZENITH LAPTOPS

Minisport	$1379
Supersport Mod 2	$1525
Supersport Mod 20	$2265
Supersport 286 Mod 20/40	$2695/2995

DISK DRIVES

SEAGATE 20 Mb Kit	$259
SEAGATE 30 Mb Kit	$279
SEAGATE 40Mb ST 251-1	$360
SEAGATE 80 Mb	$569
TOSHIBA 1.44 Mb Floppy	$110
PROCOM Ext 5.25" FOR PS/2	$279
PROCOM 45 Mb for PS/2	$695
COMPAQ 1.44 1/3 HT	$195

Card Drives

PLUS 20 Mb/40 Mb Hard Card	$535/647
PROCOM 20 Mb	$379
PROCOM 30 Mb/40 Mb	$429/519
MAUNARD 20 Mb	$Call

FAX

MURATA M1200	$Call
MURATA M1600	$899
SHARP FO220/FO330	$950/1159
SHARP FO500	$1400
TOSHIBA 3010	$1275
TOSHIBA T30100	$1049
TOSHIBA T3300	$1199
TOSHIBA T3700	$1500

COPIERS

CANON PC3-2	$539
CANON PC6RE	$1000
CANON PC7	$1210

TAPE

IBM 6157 for PS/2	$1350
IRWIN 40 Mb Int/Ext	$465
MAYNARD 60 Mb XT/At Int/Ext	$675/881
MAYNARD 60 Mb for PS/2	$899
MOUNTAIN 60 Mb	$1199

MONOCHROME MONITORS

AMDEK 410A	$145
AMDEK 432	$160
COMPAQ Mono Amber/Green	$210
COMPAQ VGA Mono	$210
IBM PS/2 8503	$200
MULTISYNC GS IIA	$239
PACKARD BELL Amber/Green	$85

COLOR/EGA MONITORS

IBM PS/2 8512	$435
PGS UltraSync	$500
SAMSUNG RGB	$245
Smartscam 735	$520
ZENITH 1390	$405

VGA/CAD MONITORS

COMPAQ AGCM Monitor	$1399
COMPAQ VGA Color	$520
IBM PS/2 8513	$515
IBM PS/2 8514	$1150
MITSUBISHI 6905	$2310
NEC MultiSync 4D/5D	$1130/2349
NEC MultiSync 2A/3D	$495/639
PACKARD BELL	$349
ZENITH 1490	$619

NOVELL NETWORK

HYUNDAI Diskless Workstation	$595
NETWARE 386	$5600

NOVELL ELS II	$1075
NOVELL 286 Software V.2.15	$1749
NOVELL SFT Level V.2.15	$2649
ARCNET PS110	$410
ARCNET PC130	$160
ARCNET PC260	$235
NE 1000 Ethernet Card	$189
NE 2000 Ethernet Card	$295
NE 232-32 bit Ethernet Card	$775
ARCNET Passive Hub	$90
ARCNET Active Hub	$469
Ethernet Terminators	$38

TERMINALS

WYSE 50 Amber or Green	$370
WYSE 60 Amber or Green	$295
WYSE 85	$394
WYSE 150	$290

SOFTWARE

Aldus Pagemaker	$480
COMPAQ DOS 3.31/4.0	$89/110
dBase IV	$449
DisplayWrite IV	$299
Harvard Graphics	$275
IBM DOS 3.3/4.01	$90/110
IBM OS/2 Standard 1.2	$255
IBM OS/2 Extended 1.2	$595
Lotus 1-2-3 V 2.2/3.0	$329
Lotus Symphony	$420
MicroSoft Windows 286/386	$73/135
MicroSoft Excel	$290
MicroSoft Word/Work	$220/110
Multimate Advantage 2	$279
Norton Advanced Utilities 4.5	$85
Paradox 3.0	$439
PFS First Pub	$75
PFS 1st Choice	$99
PFS Prowrite/File	$135/199
Q&A	$230
Quicken	$37
Rightwriter	$59
Smartcom II/III	$90/150
Ventura Publishing	$515
Word Perfect 5.0/5.1	$219/245

LASER PRINTERS

HP DeskJet Plus/IIP	$675/990
HP LaserJet II/IID with toner	$1649/2725
HP ScanJet Plus with/IF	$1475
NEC KC890	$3150
Pacific Data 25-in-1 Font	$282
Pacific Data 1-2-4	$230
Pacific Data Postscript Cart	$495
PANASONIC KXP4420	$Call
PANASONIC KXP4450	$1359
TOSHIBA PageLaser 12-B	$2499

PLOTTERS/SCANNERS, ETC.

CalComp	$Call
HP Plotter 7440A/7475A	$965/1399
HP Plotter 7550	$2920
Sumasketch 12x12/12x18	$375/660

PRINTERS

Diconix	
150 Plus	$325
300	$359
Epson	
FX 850 260/84 cps	$329
FX 1050 260/84 cps	$449
LQ 510 60/180 cps	$330
LQ 850 264/88 cps	$530
LQ 1050 264/88 cps	$740
LQ 2550 400/133 cps	$929
LX 810 30/180 cps	$195
DFX 5000 533/80 cps	$1399
IBM	
Quickwriter	$1120
Quiewriter III	$1125
Proprinter X24e	$615
Proprinter XL 24e	$797
Proprinter III XL	$645
NEC	
P2200XE	$349
P5200	$507
P5300	$695
Okidata	
ML 172P	$210
ML 182 Turbo	$230
320/21	$365/477
390/91	$464/635
393/393 Color	$980/1089
Panasonic	
KX 1180 I 192/38 cps	$179
KX 1191 I 240/48 cps	$235
KX 1124 i92/48 cps	$299
KX 1624 NEW!	$449
KX 1695 NEW!	$Call
KX 1524 240/80 cps	$540
Toshiba	
Expresswriter 301/311	$333/410

COMM./EMULATION

AST PC XT/AT 5250 Emulation	$575
EVEREX 1200 w/SW	$75
EVEREX 2400 Int/Ext	$137/195
HAYES 1200B Int/SCII	$243/282
HAYES 1200 Ext	$277
HAYES 2400 Int/Ext	$269/349
IBM PS/2 50-80 5250 Emulation	$675
IBM PC XT/AT 5250 Emulation	$675
IBM PC XT/AT 3270 Emulation	$450
MEGAHERTZ 2400B	$169
US ROBOTICS 1200 Int	$100
US ROBOTICS 2400 Int	$159

MULTIFUNCTION CARDS

AST 6 Pack w/384K	$189
AST Advantage/2 for PS/2	$Call
COMPAQ 1 Mb	$525
COMPAQ 4 Mb	$1900
IBM i486 Add on	$3000
Intel Above Board Plus 286/512K	$395
Intel Above Board II Plus OK	$295
Intel Above Board MC32 OK	$399
Sota 286i/386si Accel	$279/495

Curier editorial

- In intervalul de timp scurs de la aparitia primului numar al revistei PC-Magazin am primit la sediul redactiei noastre nenumarate mesaje din partea cititorilor celor mai entuziasti.

- In ciuda greutatilor pe care le-am intimpinat in realizarea unei bune distributii a revistei, scrisorile si telefoanele primite ne fac sa credem ca, totusi, revista a ajuns la cei interesati in aparitia ei.

- Vom incerca, in rindurile de mai jos, sa raspundem la o parte din problemele pe care dumneavoastra ni le-ati ridicat.

- Numerosi cititori ne-au solicitat ca macar citeva pagini sa fie dedicate microcalculatoarelor pe 8 biti (familia Sinclair SPECTRUM). Tinem sa-i anuntam, cu acest prilej, ca dorinta le-a fost indeplinita incepind cu acest numar al revistei PC-Magazin.

- In ceea ce priveste achizitionarea de calculatoare personale, revista noastra publica in paginile sale, si o va face din ce in ce mai des, numeroase reclame, liste cu preturi orientative etc.

- PC-Magazin apare lunar iar abonamentele pot fi procurate, in continuare, de la sediul nostru, telefon 66.04.76.

- Reamintim tuturor celor interesati ca sintem gata sa alocam spatiul necesar deschiderii unei rubrici de "mica publicitate" (vinzari/cumparari/ oferte de servicii). Pe masura ce materialele respective vor fi primite de la dumneavoastra, ele se vor regasi in paginile revistei.

- In ce priveste achizitionarea revistei, incepind cu numarul 2 al revistei 20000 de exemplare au fost puse la dispozitia cititorilor prin intermediul retelei de librarii.

- Ne cerem scuze pentru greselile sau inexactitatile care s-au strecurat in paginile revistei. Ele se datoreaza, in primul rind, grabei cu care trebuie sa facem fata problemelor legate de tehnoredactare si tiparire.

- Incepind cu numarul 4 vom starta (in sfirsit) o rubrica de grafica, rubrica mult solicitata de cititorii nostri.

- In ceea ce priveste serialul de C, intentionam ca, odata incheiata prezentarea generala a acestuia, sa continuam cu prezentarea Standardului C (publicat in 1989) si cu diversele implementari ale acestuia (Microsoft C, Turbo C etc.).

PTECH™ *ALP*TECH™ *ALP*TECH™ *ALP*TECH™ *ALP*TECH™ *ALP*TECH™

SWISS

BECAUSE YOU HAVE THE CHOICE

NEW CASES AND PROCESSORS

A multitude of cases and processors give you that much more to choose from: slim-line, mini-tower and tower… to suit all of your needs.

Alp Technology SA, makers of the industry-standard ALPTECH range of personal computers provide you with **the Swiss choice**.
Alp Technology combines technology and reliability with service and care – to give you the best of all worlds.
Call (41) (22) 644 525, for advice to suit your computer needs.

TM

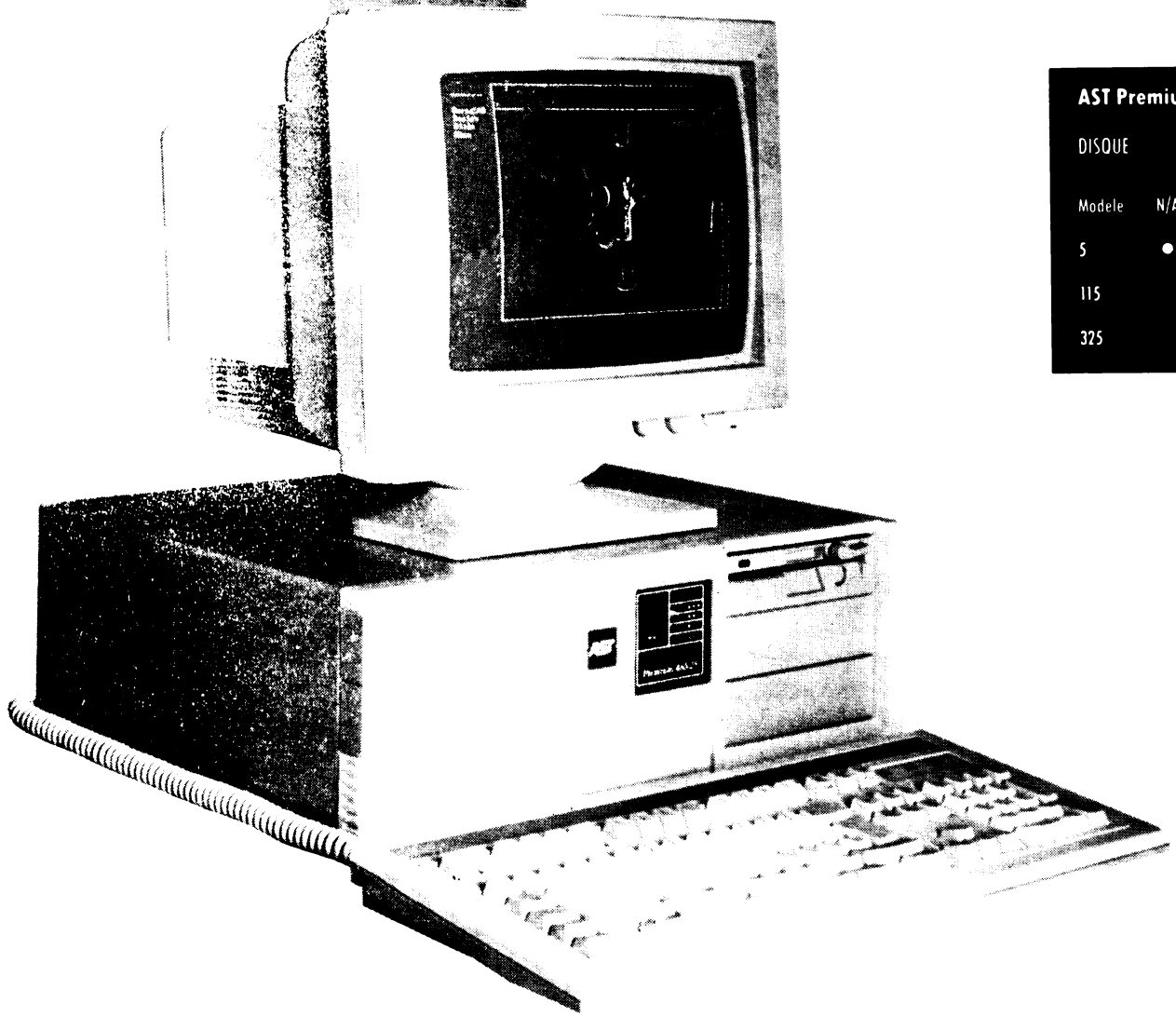
PTECH

Swiss quality comes to the world of personal computers.

Fidele et Fiable
Immer da
Trustworthy

AST Premium 486/25

La puissance d'un mini dans un compatible PC ISA



AST Premium 486/25			
DISQUE			
Modele	N/A	110 MB	320 MB
5	•		
115		•	
325			•

- Convient idéalement à toutes les applications puissantes
- Une performance digne d'une station de travail ou d'un mini
- Accepte jusqu'à 32 Mo de mémoire SIMM
- Architecture Cupid 32 pour une évolution en souplesse vers plus de puissance
- Jusqu'à 640 Mo de capacité de disques durs
- Compatibilité ISA totale
- 7 connecteurs d'extension

AST